MSX ROM-BIOS HANDBOEK

TERMINAL SOFTWARE PUBLICATIES

MSX ROM-BIOS HANDBOEK

TERMINAL

MSX

ROM-BIOS HANDBOEK

MSX

ROM-BIOS HANDBOEK

TERMINAL SOFTWARE PUBLICATIES

Een uitgave van: Terminal Software Publicaties
Postbus 111, 5110 AC Baarle Nassau

1e druk september 1986

ISBN 90-6883-026-0

Vertaald uit het Engels door P.Pauwels
Oorspronkelijke titel: The MSX Red Book.

INHOUDSOPGAVE

INLEIDING

Doelstelling van het boek

Dit boek gaat over MSX-computers en hun werking. Om technische en commerciële redenen geven de producenten van MSX-computers slechts een beperkte hoeveelheid informatie vrij over de manier waarop hun machines werken. Die informatie bestaat gewoonlijk uit een beschrijving van MSX-BASIC en wat algemeenheden over de hardware van het systeem. Die informatie is wellicht voldoende voor de gelegenheids-gebruiker van de computer, maar wanneer iemand zich aan meer gespecialiseerd programmeerwerk wil wagen, schiet ze al snel tekort.

Met de gedetailleerde informatie in dit boek willen we ook de meest veeleisende gebruiker, nl. de machinetaal-programmeur, tevreden stellen. Het boek is geen cursus programmeertechniek, en is noodzakelijkerwijs vrij technisch van opzet. We zijn er van uit gegaan, dat de lezer de werking van de Z80 processor op machinetaal-niveau al kent ofwel op een of andere manier wil leren kennen. Er bestaan al zoveel algemene boeken over de Z80, dat een beschrijving ervan zou neerkomen op een herhaling van overal beschikbare informatie.

Organisatie van het boek

De MSX-standaard schrijft als belangrijkste functionele onderdelen van elke MSX-computer, het volgende voor :

(1) Zilog Z80A microprocessor
(2) Intel 8255 programmeerbare periferie-interface (PPI)
(3) Texas 9929 video display processor (VDP)
(4) General Instruments 8910 programmeerbare geluids-generator (PSG)
(5) 32 K MSX BASIC ROM
(6) minimaal 8 K RAM

Er zijn natuurlijk nog veel meer onderdelen nodig om een MSX-computer te bouwen, maar die zijn allemaal minder grootschalig en niet programmeerbaar, waardoor ze voor de gebruiker niet zichtbaar zijn. De producenten zijn over het algemeen nogal vrij in de keuze van deze componenten. De programmeerbare delen daarentegen moeten identiek zijn; dat is de reden dat alle MSX computers in het gebruik aan elkaar gelijk zijn.

In hoofdstukken 1, 2 en 3 wordt de werking van de PPI, de VDP en de PSG beschreven. Deze drie onderdelen zorgen voor de verbinding tussen de Z80 en de randapparatuur, bij een standaard MSX-machine. Ze hebben alle drie een plaats op de In/Out-bus van de Z80 processor.

Hoofdstuk 4 bevat een beschrijving van de software in het eerste deel van de ROM. Dit deel van de ROM kontroleert tot in de details de randapparatuur van de computer, en staat bekend als het ROM BIOS (Basic Input/Output Systeem). Het is op zo'n manier gestruktureerd dat de meeste funkties die een machinetaal-programmeur nodig heeft (toetsenbord-routines, video-routines en dergelijke), gemakkelijk toegankelijk zijn.

In hoofdstuk 5 wordt de software in de rest van de ROM beschreven, namelijk de Microsoft MSX BASIC Interpreter. Hoewel dit deel van de ROM in hoofdzaak door tekst gestuurd wordt (input, programmaregels) blijkt het bij nauwkeurig onderzoek toch een aantal mogelijkheden te bevatten die niet door de fabrikanten worden vermeld.

In hoofdstuk 6 wordt ingegaan op de organisatie van het geheugen. Veel aandacht wordt besteed aan het Werkgebied ("Workspace") tussen adres F380h en FFFFh. Dat RAM-gebied gebruiken het BIOS en de BASIC Interpreter als hun eigen werkruimte. Het bevat een hoop nuttige informatie voor toepassingsprogramma's.

Hoofdstuk 7 bevat een aantal voorbeelden van machinetaal-programma's die ROM-routines gebruiken, en zo het programmeren makkelijker maken.

We denken dat in dit boek geen fouten voorkomen. Mocht u het daarmee niet eens zijn, dan wil de uitgever dat graag van u vernemen.

HOOFDSTUK 1

De programmeerbare periferie-interface (PPI)

De 8255 PPI is een algemene parallelle interface, gestruktureerd als drie 8-bits data poorten (A, B en C genoemd) met een "mode"-poort. De Z80 ziet de interface als vier In/Out-poorten waarmee het toetsenbord, de schakelapparatuur voor het geheugen, de motor van de cassetterecorder, de output naar de cassette, het lampje bij de hoofdlettertoets en de klik bij het indrukken van een toets, gestuurd kunnen worden. Nadat de PPI geïnitialiseerd werd, verloopt de toegang tot een bepaald deel van de hardware via een lees- of schrijfopdracht naar de betrokken poort.

PPI-poort A (In/Out-poort A8h)

bits	7 6	5 4	3 2	1 0
	blz. 3 PSLOT#	blz. 2 PSLOT#	blz. 1 PSLOT#	blz. 0 PSLOT#

Fig. 1 : Primair "slot"-register

Deze output-poort wordt in MSX-terminologie het "Primaire Slot-register" genoemd. Ze stuurt de hardware die nodig is bij het schakelen van het geheugen. De Z80 kan slechts 64 K geheugen rechtstreeks adresseren. Dit wordt tegenwoordig als te beperkt ervaren, en vele recente microcomputers omzeilen dit op een of andere manier.

In MSX-machines kunnen verscheidene geheugen-banken op hetzelfde adres staan. De Z80 kan ze dan op het gepaste ogenblik inschakelen of inactief maken. De geheugenruimte van de processor wordt beschouwd als zijdelings georganiseerd in vier afzonderlijke blokken van 64 K. Die blokken heten dan "Primaire slots 0 tot 3". De vier blokken ontvangen hun eigen "select" signaal naast de gewone Z80-bus signalen. De inhoud van het Primaire Slot-register bepaalt dan welk "select" signaal gegeven wordt, en daardoor ook welk Primair Slot actief is.

Om soepeler te kunnen werken, kan elke bladzijde (16 K) van de adresruimte van de Z80, aangesproken worden vanuit een verschillend "Primair Slot". In figuur 1 wordt duidelijk dat 2 bits van het PSR (Primair Slot Register) nodig zijn om voor elke bladzijde het overeenkomstig nummer van een primair slot te bepalen.

Het allereerste wat de MSX ROM doet, wanneer de stroom ingeschakeld wordt, is elk "slot" aftasten op zoek naar RAM op bladzijden 2 en 3 (8000h tot FFFFh). Het PSR wordt dan zodanig gezet, dat de betreffende "slots" actief zijn. Op die manier wordt de RAM dus permanent beschikbaar. De geheugenindeling van elke MSX-machine kan bepaald worden aan de hand van de inhoud van het PSR. Dit kan gebeuren via het Basic commando :

```
PRINT RIGHT$("0000000"+BIN$(INP(&HA8)),8)
```

Bij een Toshiba HX10 bv. zou dit als resultaat opleveren : "10100000" : bladzijden 3 en 2 (RAM) komen beide van slot 2, en bladzijden 1 en 0 (MSX ROM) beide van slot 0. De MSX-ROM moet door de fabrikant altijd in slot 0 geplaatst worden : bij aanschakelen wordt dit slot geselecteerd door de hardware. Andere geheugenstrukturen, RAM of bijkomende ROM, kunnen om het even waar een plaats krijgen.

Een gemiddelde MSX-machine heeft een Primair Slot met de MSX-ROM, één met 64 K RAM, en twee in verbinding met aansluitingen naar buiten de computer. De meeste Japanse machines hebben een aansluitbus voor cartridges op elk van deze externe aansluitingen; andere machines daarentegen hebben gewoonlijk slechts één cartridge-aansluiting plus een IDC-connector.

Uitbreidingen

Het geheugen van de computer kan uitgebreid worden tot een theoretisch maximum van 16 blokken van 64 K, middels uitbreidings-interfaces. De gebruiker kan een interface aansluiten op elke Primair Slot, waardoor hij vier Secundaire Slots van 64 K krijgt, genummerd van 0 tot 3, in plaats van één Primair Slot. Elke uitbreidings-interface heeft zijn eigen hardware aan boord (het Secundaire Slot Register of SSR), om te bepalen welk Secundair Slot in het Primaire Slot aanwezig dient te zijn. Geheugenbladzijden kunnen ook hier vanuit verschillende Secundaire Slots geselecteerd worden.

bits 7 6	5 4	3 2	1 0
blz. 3 SSLOT#	blz. 2 SSLOT#	blz. 1 SSLOT#	blz. 0 SSLOT#

Fig. 2 : Secundair Slot Register

Elk SSR is in feite een 8 bits lees/schrijf geheugen. De hardware van de uitbreidingsinterface zorgt ervoor dat het als adres FFFFh van zijn PSR gezien wordt. Om dit adres in een bepaald uitbreidingsblok te bereiken, zal de gebruiker meestal eerst bladzijde 3 (C000h tot FFFFh) van het betrokken Primaire Slot in het geheugenbereik van de processor moeten schakelen, waarna het SSR gewijzigd kan worden en, zo nodig, terug geschakeld kan worden naar bladzijde 3 van het oorspronkelijke Primaire Slot. De toegang tot geheugenplaatsen in uitbreidings-blokken kan dus een vrij omslachtig gebeuren worden.

Het is duidelijk dat op een of andere manier moet kunnen bepaald worden of een Primair Slot gewone RAM bevat, dan wel een uitbreidingsblok, om het op de goede manier te kunnen aanspreken. Daartoe werden de SSR's zo geconstrueerd, dat ze hun inhoud inverteren, wanneer ze uitgelezen worden. Bij het zoeken naar RAM, na het inschakelen van de machine, wordt adres FFFFh

van elk Primair Slot bekeken, en wordt vastgesteld of het zich normaal gedraagt dan wel of er een uitbreidingsblok in zit. Het resultaat van dit onderzoek wordt opgeslagen in het Werkgebied, in de tabel EXPTBL, voor later gebruik. Dit gebeurt bij het inschakelen, omdat het uitvoeren van tests moeilijker is wanneer de SSR's al relevante informatie bevatten.

Het schakelen van geheugen vraagt uiteraard de grootste voorzichtigheid, zeker in verband met het hiërarchische systeem dat nodig is om de uitbreidingsblokken te controleren. Schakel nooit een bladzijde uit waarin een programma loopt of waarin zich de stack bevindt, als die gebruikt wordt. In het BIOS-gedeelte van de ROM vindt de machinetaal-programmeur een aantal standaard-routines die hem het leven wat gemakkelijker maken.

De BASIC Interpreter kan zelf op vier manieren uitbreidings-ROMS aanspreken. De eerste drie daarvan worden gebruikt bij machinetaal-ROMS op bladzijde 1 (4000h tot 7FFFh). Die manieren zijn :

1) de "hooks" (zie hoofdstuk 6)
2) het "CALL"-statement (zie hoofdstuk 5)
3) extra namen voor randapparatuur (zie hoofdstuk 5)

De BASIC Interpreter kan ook een BASIC programma in ROM uitvoeren. Dat programma moet tijdens de initialiserings-routine gevonden worden op bladzijde 2 van het geheugen (8000h tot BFFFh). De Interpreter heeft geen toegang tot RAM die verborgen ligt achter andere geheugenstrukturen. Die beperking maakt het moeilijk om een bestaand programma zodanig te herschrijven, dat het gebruik kan maken van de mogelijkheden van nieuwere en complexere machines. Daar heeft ook de versie van Microsoft BASIC voor de IBM PC last van : van het 1 MB geheugen kan slechts 64 K voor programmatuur gebruikt worden.

PPI Poort B (In/Out-poort A9h)

7	6	5	4	3	2	1	0
Inputs van de kolommen van het toetsenbord							

Fig. 3

Deze input-poort wordt gebruikt om de 8 bits koloms-informatie te lezen van de op dat ogenblik afgetaste rij op het toetsenbord. Het toetsenbord van de MSX wordt softwarematig afgetast. Het is gestruktureerd als een matrix van elf rijen bij acht kolommen van schakelaars, die open zijn wanneer ze niet bediend worden. De huidige machines bezetten met hun toetsen de rijen 0 tot 8. De "vertaling" van een toetsindruk naar codes wordt door de interrupt-routines in de MSX-ROM verzorgd. Lees hoofdstuk 4 voor nadere uitleg daarover.

PPI Poort C (Input/Output poort AAh)

7	6	5	4	3	2	1	0
toets klik	CAPS lamp	cass. outp.	cass. motor	selectie van de rij op het toetsenbord			

Fig. 4

Deze output-poort heeft een aantal funkties.

De vier bits 0 tot 3 bepalen welke rij op het toetsenbord (0 tot 10) door poort B ingelezen moet worden.

Bit 4, het cassettemotor-bit, bepaalt de stand van het relais dat de cassettemotor schakelt. Aan = 0, uit = 1.

Bit 5, het cassette-output bit, wordt gefilterd en verzwakt vooraleer het naar de DIN-plug gaat, als het signaal voor de cassette-recorder. Alle cassettesignalen worden via software gegenereerd.

Bit 6, het CAPS-bit, bepaalt of het lampje bij de hoofdlettertoets aan (0) of uit (1) is.

Bit 7 bepaalt het geluid dat gemaakt wordt bij het indrukken van een toets. Het wordt verzwakt en gemengd met het signaal van de geluidsgenerator. Om een geluid voort te brengen, moet dit bit voortdurend tussen 0 en 1 heen en weer gezet worden.

In het ROM BIOS staan standaard-routines die alle funkties van deze poort bruikbaar maken. Het is beter om die routines te gebruiken dan om rechtstreeks de hardware te manipuleren - als dit al mogelijk is.

PPI "mode" poort (Input/Output poort ABh)

7	6	5	4	3	2	1	0
1	A en C mode		A dir.	C dir.	B en C mode	B dir.	C dir.

Fig. 5 : Instelling van de PPI-mode

Deze poort regelt de werking van de PPI. De MSX-hardware werd ontworpen om in een bepaalde configuratie te werken; wijzig de inhoud van deze poort dus nooit. De informatie hieronder wordt enkel gegeven uit oogpunt van volledigheid.

Bit 7 moet 1 zijn om de "mode" waarin de PPI werkt, te wijzigen. Als dit bit 0 is, wordt een bit van poort C geset of gereset (zie fig. 6).

De "A&C mode"-bits bepalen de werking van poort A, en de 4 hoogste bits van poort C : 00 - normale werking (MSX), 01 - strobe mode, 10 - tweerichtingsverkeer.

Het "A direction"-bit bepaalt in welke richting poort A werkt : 0 is output (MSX), 1 is input.

Het "C direction"-bit bepaalt in welke richting de hoogste vier bits van poort C werken : 0 is output (MSX), 1 is input.

Het "B&C mode"-bit stuurt de manier waarop poort B werkt en de laagste vier bits van poort C : 0 is normale werking (MSX), 1 is strobe mode.

Het "B direction"-bit geeft de richting aan voor poort B : 0 is output, 1 is input (MSX).

Het "C direction"-bit bepaalt in welke richting de laagste vier bits van poort C werken : 0 is output (MSX), 1 is input.

7	6 5 4	3 2 1	0
0	niet gebruikt	nummer vh bit	set/reset

Fig. 6 : De bit set/reset-mogelijkheid van de PPI

Deze poort kan gebruikt worden om rechtstreeks een bit van poort C te setten of te resetten, wanneer bit 7 van de poort, nul is. Het bit-nummer, van 0 tot 7, bepaalt om welk bit het gaat. De waarde die het aangegeven bit moet aannemen, wordt bepaald door het set/reset bit (bit 0) : set - 1, reset - 0. Het voordeel hiervan is, dat één enkele output gemakkelijk te wijzigen is. Het hoofdletter-lampje kan bijvoorbeeld aangezet worden met de Basic instructie OUT &HAB,12 en weer uitgezet door de instructie OUT &HAB,13.

Bit 7 moet 1 zijn om de "mode" definitie te [illegible] worden, en [illegible]. Als dit bit 0 is, wordt een bit van poort C gezet of gereset (zie Fig. 6).

De "ABC mode" bits bepalen de werking van poort A en de 4 hoogste bits van poort C. 00 = normale werking (mode 0), 01 = strobe mode, 1X = twee-richtingsverkeer.

Het "A directioneel" bit bepaalt de werkingsrichting van poort A waarbij 0 is output (OUT), 1 is input.

Het "C directioneel" bit bepaalt de werkingsrichting de hoogste vier bits van poort C waarbij 0 is output (OUT), 1 is input.

Het "BC mode" bit bepaalt de werking van poort B en de laagste vier bits van poort C. 0 is normale werking (mode 0), 1 is strobe mode.

Het "B directioneel" bit geeft de richting van poort B aan: 0 is output, 1 is input (IN).

Het "C directioneel" bit bepaalt de werkingsrichting de laagste vier bits van poort C waarbij 0 is output (OUT), 1 is input.

7	6	5	4	3	2	1	0
0	niet gebruikt			nummer v. bit			set/reset

Fig. 6 : De bit set/reset control word van de PPI

Deze poort kan gebruikt worden om afzonderlijke bits van poort C te setten of te resetten, wanneer bit 7 van de control word 0 is. Het bit-nummer, van 0 tot 7, bepaalt om welk bit het gaat. De waarde die het aangegeven bit moet aannemen, wordt bepaald door het set/reset bit (dat 0 = reset, 1 = set). Het voordeel hiervan is, dat dan enkele output [illegible] kunnen worden. Het hoofdletter-lampje van [illegible] aangezet worden met de Basic instructie [illegible] wordt [illegible] door de instructie OUT &HAB,13

HOOFDSTUK 2

De video display processor (VDP)

De 9929 chip bevat alle elementen die nodig zijn voor de opbouw van de video display, het beeldscherm. De Z80 ziet de chip als twee In/Out-poorten, de Data Poort en de Commando-poort. De scherm-inhoud wordt bepaald door de 16 K Video RAM (VRAM) van de 9929 chip. De Z80 heeft geen rechtstreekse toegang tot deze RAM. Hij moet de twee In/Out-poorten gebruiken om de VRAM te wijzigen en om de diverse instellingen van de VDP uit te voeren.

Data Poort (In/Out-poort 98h)

De Datapoort wordt gebruikt om een byte naar VRAM te schrijven of eruit te lezen. De VDP heeft zijn eigen adresregister, dat VRAM-adressen bevat. Uitlezen van de Datapoort levert de inhoud op van het VRAM-adres dat in het VDP-adresregister staat. Een schrijf-bewerking naar de Datapoort zal een byte naar datzelfde adres schrijven. Na een lees- of schrijfoperatie wordt het adresregister automatisch opgehoogd, en wijst naar de volgende locatie in VRAM. De toegang tot opeenvolgende adressen in VRAM verloopt via opeenvolgende lees- of schrijfoperaties naar de Datapoort.

Commando-poort (In/Out-poort 99h)

De commandopoort wordt voor drie doeleinden gebruikt :

1) om het adresregister van de datapoort te laden
2) om het Status-register van de VDP uit te lezen
3) om naar één van de VDP-Mode registers te schrijven

Adres-register

Het adresregister van de Datapoort moet verschillend geladen worden, al naargelang de volgende operatie een lees- of schrijf-bewerking is. Het adresregister kan met elke waarde tussen 0000h en 3FFFh geladen worden, door eerst het LSB (Least Significant Byte) en dan het MSB (Most Significant Byte) naar de commandopoort te schrijven. Bits 6 en 7 van het MSB worden door de VDP gebruikt om uit te maken of het adresregister klaargemaakt moet worden voor een lees- of schrijfoperatie, en wel als volgt :

```
    lezen :  xxxxxxxx  00xxxxxx
 schrijven : xxxxxxxx  01xxxxxx
```

Fig. 7 : samenstelling van een VDP-adres

Het is van belang, dat er tussen het schrijven van het LSB en het MSB geen toegang tot de VDP meer gezocht wordt, omdat dit de synchronisatie in de war kan brengen. De interrupt-routine van de MSX ROM leest voortdurend het VDP Status register uit, zodat voor de voornoemde bewerking de interrupts afgezet dienen te worden.

VDP Status Register

Door de commandopoort uit te lezen, kennen we de inhoud van het Statusregister van de VDP. Dit register bevat ondermeer een aantal vlaggen :

7	6	5	4 3 2 1 0
F vlag	SS vlag	C vlag	Nr. van de 5de sprite

Fig. 8 : statusregister van de VDP

Bits 0 tot 4 bevatten het nummer (0 tot 31) van de sprite die de 5S-vlag actief maakt.

De C-vlag (coïncidentievlag) is normaal 0 maar wordt 1 wanneer sprites een of meer overlappende pixels hebben. Het uitlezen van het statusregister reset deze vlag. Denk eraan dat overlapping alleen gecontroleerd wordt bij het genereren van elke pixel tijdens het schrijven van een tv-beeld, dat wil zeggen elke 20 msec in Europa. Wanneer snelle sprites elkaar kruisen tussen twee zulke controles, blijft de vlag onveranderd.

De S-vlag (5de Sprite vlag) is normaal 0 maar wordt 1 wanneer er meer dan 4 sprites op een willekeurige pixel-lijn staan. Deze vlag wordt eveneens gereset door het uitlezen van het register.

De F-vlag (Frame-vlag) is normaal 0 maar wordt 1 op het einde van de laatste actieve video-lijn. In Europa (netfrekwentie 50 Hz) gebeurt dit elke 20 msec. Uitlezen van het Statusregister reset ook deze vlag. Daarmee samenhangend geeft de VDP een interrupt-signaal aan de Z80, met dezelfde frekwentie, waardoor de interrupt-routines van de MSX ROM gestuurd worden.

De Mode registers van de VDP

De VDP bevat 8 registers (0 tot 7) waarin enkel geschreven kan worden, en die de algemene werking ervan besturen. De inhoud van een bepaald register wordt gewijzigd door eerst een data-byte, en vervolgens het registernummer naar de commandopoort te schrijven. Dit laatste byte bevat het registernummer in de laagste drie bits. Deze mode-registers kunnen enkel beschreven worden, en niet uitgelezen : daarom zet de MSX-ROM een kopie van de inhoud ervan in het Werkgebied (zie hoofdstuk 6). Om zeker te zijn dat alles correct verloopt, is het derhalve beter om de

standaard routines in de ROM te gebruiken voor VDP-functies.

Mode register 0

7	6	5	4	3	2	1	0
0	0	0	0	0	0	M3	EV

Fig. 9

Bit 0, het Externe Video bit, bepaalt of externe video-input al dan niet toegelaten is. Niet toegelaten = 0; wel toegelaten = 1.

Bit 1, het M3 bit is een van de drie bits waarmee de VDP-mode ingesteld kan worden : zie mode register 1.

Mode register 1

7	6	5	4	3	2	1	0
4 of 16K	wis	IE	M1	M2	0	maat	magn

Fig. 10

Bit 0 (magnification bit, vergrotingsbit) bepaalt of sprites op normale grootte dan wel vergroot worden afgebeeld : 0 = normaal, 1 = dubbele grootte.

Bit 1 (maat bit) bepaalt of een sprite-patroon 8x8 bits (0) of 16x16 bits (1) groot is.

Bits 2 en 3 samen met bit 1 van register 0 bepalen de mode waarin de VDP werkt :

M1	M2	M3	
0	0	0	32 x 24 tekst-mode
0	0	1	grafisch scherm
0	1	0	veelkleurig scherm
1	0	0	40 x 24 tekst-mode

Bit 5 (Interrupt Enable) bepaalt of het interruptsignaal van de VDP naar de Z80 gestuurd wordt (bit 5 = 1) of niet (bit 5 = 0).

Bit 6 (Blank-bit, wis-bit) stuurt de output naar het gehele scherm : is het 0 dan is het scherm inactief, en krijgt het dezelfde kleur als de border; is het bit 1 dan werkt het scherm gewoon.

Bit 7 past de manier waarop de VRAM geadresseerd wordt, aan 4 K of 16 K-chips aan : 0 = 4 K, 1 = 16 K.

Mode register 2

7	6	5	4	3	2	1	0
0	0	0	0	basis vr naam-tabel			

Fig. 11

Mode register 2 bevat het adres van de Namentabel in de VRAM. De vier bits die daarvoor beschikbaar zijn, geven enkel bits 2 tot 5 aan van het LSB van het volledige adres. Wanneer het register bv. 0Fh bevat, geeft dit dus een basis-adres van 3C00h.

Mode register 3

7	6	5	4	3	2	1	0
basis voor kleurentabel							

Fig. 12

Register 3 geeft het startadres van de kleurentabel in VRAM. De 8 beschikbare bits slaan op bits 0 tot 6 van het LSB en bits 6 en 7 van het MSB van het volledige adres. Een register met inhoud FFh geeft dus een basisadres 3FC0h. In scherm 1 (grafisch scherm) speelt enkel bit 7 mee, waardoor de basis dus 0000h of 2000h kan zijn. Bits 0 tot 6 moeten dan 1 zijn.

Mode register 4

7	6	5	4	3	2	1	0
0	0	0	0	0	basis voor het karakter-patroon		

Fig. 13

Register 4 geeft het startadres van de karakterset in VRAM. De drie beschikbare bits slaan op bits 3 tot 5 van het LSB van het adres. Inhoud 07h in dit register geeft dus adres 3800h. In grafische mode (scherm 1) is enkel bit 2 van belang, waardoor het adres dus 0000h of 2000h is. Bits 0 en 1 moeten 1 zijn.

Mode register 5

7	6	5	4	3	2	1	0
0	basis vr sprite-attributen						

Fig. 14

Register 5 bepaalt het startadres van de tabel met sprite-attributen in de VRAM. De zeven bits bepalen bit 7 van het MSB en bits 0 tot 5 van het LSB van het adres. Als het register dus 7Fh bevat, is het adres 3F80h.

Mode register 6

7	6	5	4	3	2	1	0
0	0	0	0	0	basis vr het sprite-patroon		

Fig. 15

Mode register 6 definieert het startadres van de tabel met sprite-patronen in VRAM. De drie bits slaan op bits 3 tot 5 van het LSB van het adres. Als dit register 07h bevat, is het basisadres dus 3800h.

Mode register 7

7	6	5	4	3	2	1	0
tekstkleur 1				border-kleur			

Fig. 16

Bits 0 tot 3 bepalen de kleur rond het actieve video-gedeelte in alle vier de VDP-modes. Ze geven ook de kleur aan voor de niet gesette pixels op het scherm (achtergrondkleur) in 40 x 24 tekst mode. Eigenlijk is het zo, dat de border over het hele scherm loopt, maar alleen zichtbaar wordt binnen het actieve video-deel indien een pixel op dit deel transparant is.

Bits 4 tot 7 bepalen de kleur van alle gesette pixels in 40 x 24 tekst mode. Ze hebben geen uitwerking in de andere drie VDP-modes waar meer mogelijkheden voorhanden zijn, door het gebruik van de kleurentabel.

De kleurcodes van de VDP zijn :

0 transparant	4 donkerblauw	8 rood	12 donkergroen
1 zwart	5 lichtblauw	9 helderrood	13 paars
2 groen	6 donkerrood	10 geel	14 grijs
3 lichtgroen	7 hemelsblauw	11 lichtgeel	15 wit

Scherm-indelingen

De VDP kan op vier manieren werken. Elke manier biedt zijn eigen mogelijkheden. Het is over het algemeen zo dat hoe hoger de resolutie wordt, hoe meer VRAM het scherm in beslag neemt, en hoe ingewikkelder de scherm-organisatie wordt. Bij een andere toepassing van deze chip, kunnen dit belangrijke overwegingen zijn. Voor een MSX-machine gaan ze evenwel niet op, en het is daarom jammer dat er niet meer geprobeerd werd om een bepaalde scherm-mode te standaardiseren. Op enkele kleine uitzonderingen na kan de grafische mode alle functies aan van de andere modes.

Een bijkomende moeilijkheid bij het gebruik van de VDP is dat er niet voldoende rekening werd gehouden met het feit dat de meeste tv-apparaten een te breed beeld geven. Dat brengt met zich mee dat karakters aan de rand van het beeld verloren gaan; daarom gaat alle ROM-software van de MSX, die met het display-gedeelte te maken heeft, uit van nogal vreemde scherm-afmetingen. Engelse machines gebruiken de middelste 37 karakters van het 40 x 24 scherm; Japanse machines met hun NTSC-standaard (National Television Standards Committee), gebruiken er 39.

Voor de programmeur is de Namentabel het centrale element in de VDP. Die tabel is gewoon een lijst met karaktercodes van 1 byte lang, in VRAM. In 40 x 24 tekstmode is die tabel 960 bytes lang. In 32 x 24 tekst mode, grafische mode en veelkleuren-mode is ze 768 bytes lang. Elke plaats in de Namentabel komt overeen met een bepaalde plaats op het scherm.

Terwijl een tv-beeld wordt opgebouwd, leest de VDP achtereenvolgens elke karaktercode uit die tabel, te beginnen bij de basis. Na het lezen van een karaktercode wordt het overeenkomstig 8x8 bitpatroon opgezocht in de tabel met karakterpatronen, en vervolgens op het scherm gezet. Hoe het scherm er uit ziet, kan dus beïnvloed worden door de karaktercodes in de Namentabel te wijzigen, ofwel door de pixelpatronen in de Karaktertabel te veranderen.

Merk op dat de VDP geen ingebouwde mogelijkheid heeft om een cursor zichtbaar te maken. Wanneer we een cursor willen, moeten we die maken via software.

40 x 24 tekst-mode

De Namentabel beslaat 960 bytes VRAM, vanaf 0000h tot 03BFh, volgens het volgende patroon :

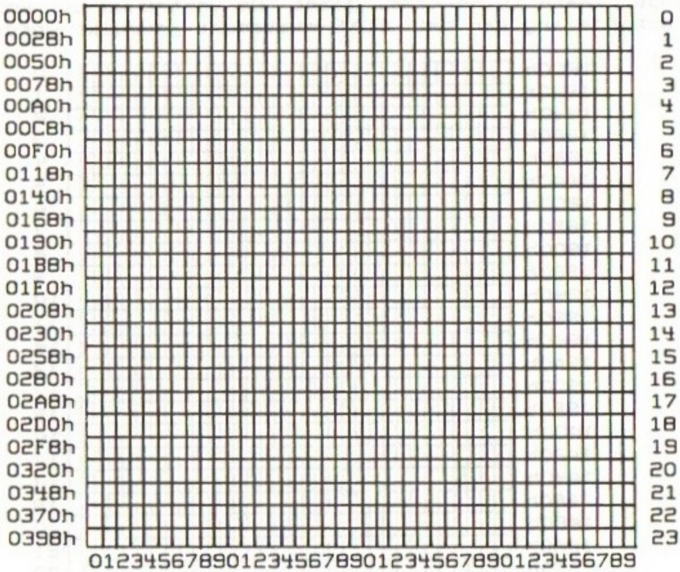


Fig. 17 : Namentabel 40 x 24

De Karaktertabel beslaat 2 K VRAM vanaf adres 0800h tot 0FFFh. Elk blok van 8 bytes bevat het pixelpatroon voor een bepaalde karaktercode :

```
00100000   byte 0
01010000   byte 1
10001000   byte 2
10001000   byte 3
11111000   byte 4
10001000   byte 5
10001000   byte 6
00000000   byte 7
```

Fig. 18 : blok met een karakter-patroon (code 65)

Het eerste blok bevat het patroon voor het karakter met de code 0; dan volgt het blok voor code 1, en zo verder tot en met 255. Enkel de 6 meest linkse pixels worden op het scherm getoond in deze mode. De kleuren van de gesette en geresette pixels (1 en 0 respectievelijk) worden bepaald door het Mode register 7. Bij het aanzetten zijn die wit en blauw .

32 x 24 mode

De Namentabel beslaat 768 bytes VRAM, van 1800h tot 1AFFh. Zoals in de 40 x 24 mode, worden in deze mode ook karaktercodes op de geëigende plaats in de tabel gezet. Om vertrouwd te worden met de schermindeling kan het VPOKE-statement gebruikt worden :

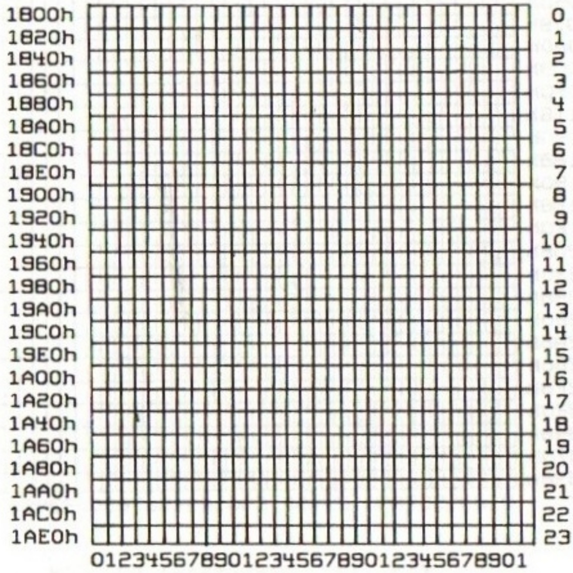


Fig. 19 : Namentabel 32 x 24 karakters

De Karaktertabel is 2 K lang, en loopt vanaf 0000h tot 07FFh in VRAM. De structuur ervan is dezelfde als in 40 x 24 mode, maar nu worden alle 8 pixels op het scherm getoond.

De kleur van de border wordt bepaald door moderegister 7, en is blauw bij het aanzetten. In een bijkomende tabel, de Kleurentabel, worden de kleuren van de pixels bijgehouden. Deze tabel is 32 bytes lang en loopt van 2000h tot 201Fh in VRAM. Elke byte in deze tabel bepaalt de kleur van een "1" en "0" pixel voor een groep van acht karaktercodes. Bits 0 tot 3 bepalen de achtergrondkleur, bits 4 tot 7 de voorgrondkleur. Byte 1 in de tabel bevat de kleuren voor de karakters met code 0 tot 7; het tweede byte bepaalt de karakters van 8 tot 15, en zo verder voor de 32 bytes in de tabel. Bij het aanzetten wordt de hele tabel geïnitialiseerd met de waarde voor blauw en wit. De ROM voorziet geen mogelijkheid om individele bytes te wijzigen.

Grafische mode

De Namentabel beslaat 768 bytes in VRAM, van 1800h tot 1AFFh, net zoals in 32 x 24 tekst-mode. De tabel wordt bij het aanzetten opgevuld met de codes 0 tot 255, drie keer na elkaar, en verder wordt die reeks niet meer gewijzigd : in grafische mode wordt de Karaktertabel gewijzigd tijdens het werken.

De Karaktertabel is 6 K lang, van 0000h tot 17FFh in VRAM. Hoewel de structuur van deze tabel dezelfde is als in tekstmode, bevat hij geen karakterset, maar enkel nullen. De eerste 2 K van de Karaktertabel wordt geadresseerd door het eerste derde van de Namentabel; de volgende 2 K door het tweede derde ervan, en het derde blok van 2 K door het laatste deel van de Namentabel. Omwille van het sequentieel karakter van de Namentabel, wordt tijdens de opbouw van een tv-beeld de hele Karaktertabel lineair uitgelezen. Een punt op het scherm zetten betekent dus : berekenen waar in de Karaktertabel het overeenkomstige bit staat, en dit dan setten. In hoofdstuk 4 staat de MAPXYC standaardroutine om XY-coördinaten om te rekenen in een adres.

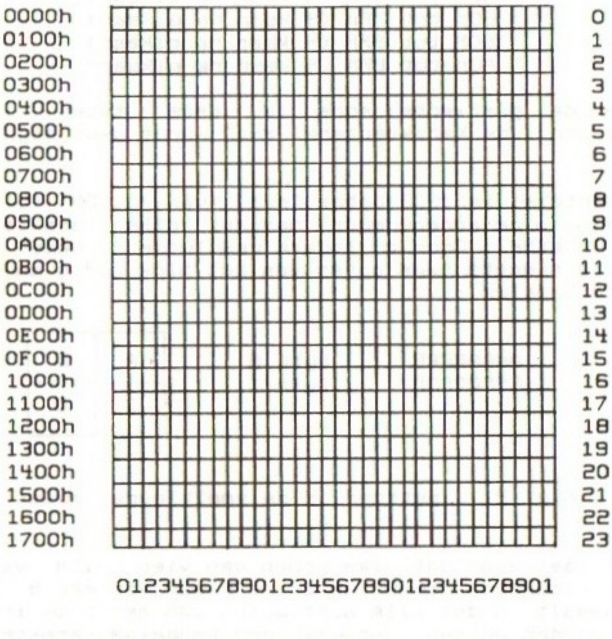


Fig. 20 : tabel van de grafische karakterpatronen

De borderkleur wordt bepaald door het moderegister 7, en is blauw bij het inschakelen. De Kleurentabel is 6 K, van 2000h tot 37FFh in VRAM. De lengte van de tabel met karakterpatronen is exact gelijk aan die van de Kleurentabel, maar het vergt 8

bits om de kleuren van de "1" en "0"-pixels op te slaan. Daardoor is er voor kleuren een lagere resolutie dan voor pixels. Bits 0 tot 3 van een byte in de Kleurentabel bepalen de kleur van de "0"-pixels van de overeenkomstige 8-pixel-lijn op het scherm. De overige vier bits bepalen de kleur van de gesette pixels. Bij het initialiseren wordt de tabel opgevuld met de waarde voor blauw voor alle pixels. Daarom zal, wanneer een bit geset wordt in de Karaktertabel, één van de kleuren gewijzigd moeten worden, hetzij de kleur voor de "0"-pixels, hetzij die voor de "1"-pixels.

<u>Veelkleuren-mode</u>

De Namentabel beslaat 768 bytes VRAM, van 0800h tot 0AFFh. De schermindeling is zoals in de 32 x 24 tekst-mode. De tabel wordt oorspronkelijk opgevuld met het volgende karakterpatroon :

00h tot 1Fh (4 keer na elkaar)
20h tot 3Fh (4 keer na elkaar)
40h tot 5Fh (4 keer na elkaar)
60h tot 7Fh (4 keer na elkaar)
80h tot 9Fh (4 keer na elkaar)
A0h tot BFh (4 keer na elkaar)

Zoals in de grafische mode is deze tabel louter een pilootpatroon : de Karaktertabel zelf wordt gewijzigd tijdens de werking.

De Karaktertabel is 1536 bytes lang, van 0000h tot 05FFh in VRAM. Zoals in de andere modes beslaat elke karaktercode acht bytes in de tabel. Door de lagere resolutie in deze mode, zijn er in feite slechts twee bytes van het blok nodig om een 8 x 8 patroon te bepalen :

AAAABBBB Byte 0
CCCCDDDD Byte 1

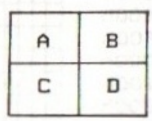


Fig. 21 : patroon in de veelkleuren-mode

Figuur 21 laat zien dat elke groep van vier bits van de twee bytes een kleurcode bevat, en zo de kleur van een 8 x 8 pixel blokje bepaalt. Opdat alle acht bytes van een blok in de tabel gebruikt zouden worden, verwijst een bepaalde karaktercode naar een verschillende groep van twee bytes, afhankelijk van de plaats die die karaktercode op het scherm inneemt (dat wil zeggen de plaats ervan in de Namentabel) :

videolijnen 0,4,8,12,16,20 gebruiken bytes 0 en 1
videolijnen 1,5,9,13,17,21 gebruiken bytes 2 en 3
videolijnen 2,6,10,14,18,22 gebruiken bytes 4 en 5
videolijnen 3,7,11,15,19,23 gebruiken bytes 6 en 7

Wanneer de Namentabel opgevuld werd met de speciale piloot-groep van karaktercodes zoals reeds vermeld, zal de Karaktertabel lineair worden uitgelezen tijdens de opbouw van een tv-beeld :

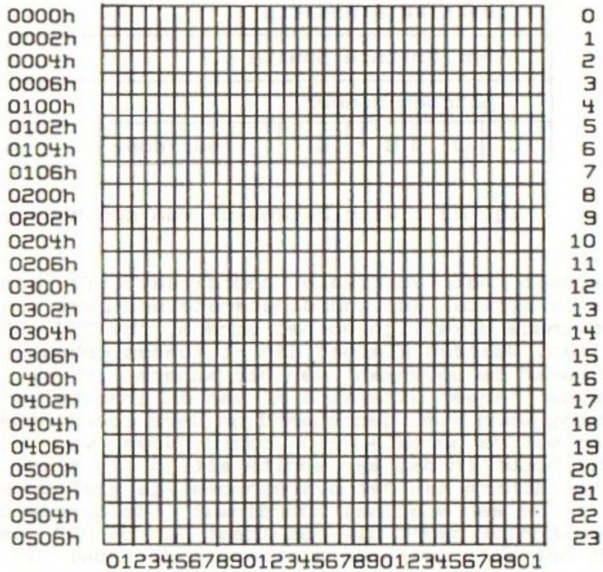


Fig. 22 : Karaktertabel in veelkleuren-mode

De kleur van de border wordt bepaald door het moderegister 7, en is oorspronkelijk blauw. Er is geen afzonderlijke Kleurentabel, aangezien de kleuren rechtstreeks bepaald worden door de inhoud van de Karaktertabel, die op blauw wordt geïnitialiseerd.

Sprites

De VDP kan 32 sprites sturen in alle modes behalve in de 40 x 24 tekst-mode. Ze worden in alle modes op een identieke manier behandeld, die los staat van de verwerking van alfanumerieke karakters.

De Attributentabel voor de sprites is 128 bytes lang, en loopt van 1B00h tot 1B7Fh in VRAM. De tabel bevat tweeëndertig blokjes van vier bytes, één per sprite. Het eerste blokje stuurt sprite 0 (de "bovenste"), het tweede blokje sprite 1 en zo verder tot sprite 31.

De struktuur van elk blok ziet er zo uit :

<table>
<tr><th>7</th><th>6</th><th>5</th><th>4</th><th>3</th><th>2</th><th>1</th><th>0</th><th></th></tr>
<tr><td colspan="8">vertikale positie</td><td>byte 0</td></tr>
<tr><td colspan="8">horizontale posite</td><td>byte 1</td></tr>
<tr><td colspan="8">patroon-nummer</td><td>byte 2</td></tr>
<tr><td>EC</td><td>0</td><td>0</td><td>0</td><td colspan="4">kleur-code</td><td>byte 3</td></tr>
</table>

Fig. 23 : attributenblok van een sprite

Byte 0 bepaalt het vertikale coördinaat (Y) van de pixel linksboven de sprite. Het coördinatenstelsel gaat van -1 (FFh) voor de bovenste pixellijn op het scherm, tot 190 (BEh) voor de onderste lijn. Waarden kleiner dan -1 kunnen gebruikt worden, om de sprite vanaf de bovenkant van het scherm in te voeren. De exacte waarden hangen uiteraard af van de grootte van de sprite. Vreemd genoeg werd niet geprobeerd om deze coördinaten te laten samenvallen met de normale 0-191 coördinaten van het grafisch scherm. Dit houdt in dat een sprite altijd 1 pixel lager zal staan op het scherm dan het overeenkomstige grafische coördinaat aangeeft. Er is een speciaal coördinaat : de waarde 208 (D0h) in een attributenblok geeft als resultaat dat de VDP geen rekening meer houdt met alle volgende blokken in de tabel. In de praktijk wil dit zeggen dat alle "onderliggende" sprites van het scherm verdwijnen.

Byte 1 bepaalt het horizontale coördinaat (X) van de pixel linksboven de sprite. De coördinaten gaan van 0 tot 255 (FFh) (voor de meest rechtse pixel). Deze coördinaten voorzien niet in een manier om de sprite van links in het scherm te voeren. Daartoe dient een bit in byte 3 (zie verderop).

Met de inhoud van byte 2 wordt één van de 255 8 x 8 bitpatronen geselecteerd in de sprite-tabel. Indien het grootte-bit in mode-register 1 geset is, wat een 16 x 16 bits-patroon oplevert dat 32 bytes beslaat, wordt met de laagste twee bits van het patroon geen rekening gehouden. Dat betekent dat patroon-nummers 0,1,2 en 3 allemaal patroon nummer 0 selecteren.

Bits 0 tot 3 van byte 3 bepalen de kleur van de gesette pixels in het sprite-patroon. Geresette pixels zijn altijd transparant. Het EC (Early Clock) bit is gewoonlijk 0, maar doet de sprite 32 pixels naar links verschuiven als het 1 wordt. Daardoor kan een sprite van linksaf in het scherm schuiven.

De spritepatroon-tabel is 2 K lang, van 3800h tot 3FFFh in VRAM. Hij bevat 256 8 x 8 pixelpatronen (0-255). Is het grootte-bit in het moderegister 1 gereset, (8 x 8 sprites) dan is de struktuur van de tabel identiek aan die van de Karaktertabel (fig. 18). Is het bit geset, dan bepalen 4 blokken van 8 bytes het patroon :

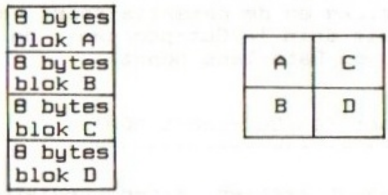


Fig. 24 : patroon voor 16 x 16 sprites

HOOFDSTUK 3

Programmeerbare geluidsgenerator

De 8910 PSG-chip stuurt drie geluidskanalen. Hij bevat ook twee acht-bits datapoorten (A en B) waarlangs hij de communicatie met de joysticks en de cassette-poort verzorgt. Voor de Z80 bestaat de PSG uit drie In/Out-poorten : de Adrespoort, de Data-schrijf-poort en de Data-lees-poort.

Adrespoort (In/Out-poort A0h)

De PSG bevat zestien eigen registers die zijn hele werking bepalen. Een bepaald register wordt geselecteerd door zijn rangnummer (0 tot 15) naar deze poort te schrijven. Na de selectie, is dit register toegankelijk via de twee datapoorten.

Data-schrijfpoort (In/Out-poort A1h)

Via deze poort wordt een register beschreven, dat voordien via de adrespoort werd geselecteerd.

Data-leespoort (In/Out-poort A2h)

Via deze poort wordt een register uitgelezen, dat voordien via de adrespoort werd geselecteerd.

Registers 0 en 1

7	6	5	4	3	2	1	0	
frekwentie kanaal A,LSB								register 0
x	x	x	x	kanaal A,MSB				register 1

Fig. 25

Met behulp van deze twee registers wordt de frekwentie bepaald van de toongenerator voor kanaal A. Diverse frekwenties worden geproduceerd door een vaste grondfrekwentie te delen door het getal in de registers 0 en 1. Dit getal kan liggen tussen 1 en 4095. In register 0 zitten de 8 laagste bits, en register 1 bevat de hoogste vier bits. De PSG deelt een externe frekwentie van 1,7897725 MHz door zestien, wat een grondfrekwentie van 111.861 Hz oplevert. De toongenerator kan dus een toon produceren tussen 111.861 Hz (deler in de registers is 1) tot 27,3 Hz (deler is 4095). Om bijvoorbeeld een gewone "A" ("la")

te horen, moeten de registers 254 bevatten : dit geeft 440 Hz.

Registers 2 en 3

Deze twee registers sturen de toongenerator voor kanaal B, op dezelfde manier als registers 0 en 1 dat voor kanaal A doen.

Registers 4 en 5

Deze twee registers sturen de toongenerator voor kanaal C, op dezelfde manier als registers 0 en 1 dat voor kanaal A doen.

Register 6

7	6	5	4 3 2 1 0
x	x	x	ruis-frekwentie

Fig. 26

Naast drie vierkantsgolf-generators bevat de PSG ook één ruisgenerator. De grondfrekwentie daarvan kan op een gelijkaardige wijze worden beïnvloed als die van de toongenerators. De vijf laagste bits van het register 6 bevatten de deler, tussen 0 en 31. De grondfrekwentie van de ruisgenerator is eveneens 111.861 Hz.

Register 7

7	6	5	4	3	2	1	0
poort B dir.	poort A dir.	ruis C	ruis B	ruis A	toon C	toon B	toon A

Fig. 27

Dit register stuurt de toongenerator en de ruisgenerator voor de drie kanalen : is een bit = 0, dan werkt de generator; is het bit = 1, dan werkt hij niet voor het betrokken kanaal. Het register stuurt ook de richting waarin poort A en B werken (de joystick en cassette-interfaces) : 0 = output, 1 = input. Register 7 moet altijd 10xxxxxx bevatten, willen we schade aan de PSG vermijden : er staat spanning op zijn aansluitingen. Het SOUND-statement zet de bits zoals het hoort, maar op machinetaal-niveau zijn we op onszelf aangewezen. Dus oppassen geblazen!

Register 8

7	6	5	4	3	2	1	0
x	x	x	mode	amplitude van kanaal A			

Fig. 28

De vier amplitude-bits bepalen de geluidssterkte van kanaal A. Minimum is 0, maximum is 15. Het mode-bit bepaalt of de sterkte konstant blijft (bit = 0) dan wel varieert (bit = 1). Wanneer dit bit 1 is, wordt geen rekening gehouden met de waarde van de vier amplitude-bits : in dit geval bepaalt de output van de envelope-generator het verloop.

Register 9

Dit register stuurt de amplitude van kanaal B, op dezelfde manier als register 8 dat doet voor kanaal A.

Register 10

Dit register stuurt de amplitude van kanaal C, op dezelfde manier als register 8 dat doet voor kanaal A.

Registers 11 en 12

7 6 5 4 3 2 1 0	
LSB van envelope-frekwentie	register 11
MSB van envelope-frekwentie	register 12

Fig. 29

Deze beide registers sturen de frekwentie van de enkele envelope-generator, die gebruikt wordt voor de modulatie van de amplitude van een kanaal. Net zoals voor de toongeneratoren, wordt de frekwentie bepaald door een deler in de registers te plaatsen. Die deler mag liggen tussen 1 en 65535, waarbij register 11 het LSB bevat en register 12 het MSB. De grondfrekwentie van de envelope-generator is 6991 Hz. De frekwentie van de envelope kan dus liggen tussen 6991 Hz (deler 1) tot 0,11 Hz (deler 65535).

Register 13

7	6	5	4	3	2	1	0
x	x	x	x	vorm vd envelope			

Fig. 30

De laagste vier bits van dit register bepalen de vorm van de envelope die de generator moet produceren :

bit 3	2	1	0
0	0	x	x
0	1	x	x
1	0	0	0
1	0	0	1
1	0	1	0
1	0	1	1
1	1	0	0
1	1	0	1
1	1	1	0
1	1	1	1

vorm van de envelope

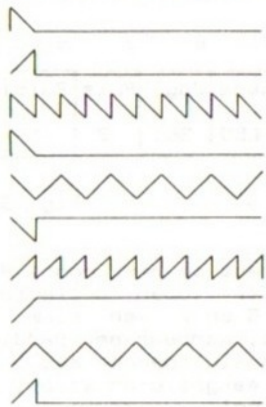


Fig. 31

Register 14

7	6	5	4	3	2	1	0
cas input	Kbd mode	Joy. Tr.B	Joy. Tr.A	Joy. Right	Joy. Left	Joy. Back	Joy. Fwd.

Fig. 32

Via dit register wordt poort A van de PSG ingelezen. De laagste zes bits, de joystick-bits, geven de stand van de vier richtings-schakelaars en de twee vuurknoppen van een joystick weer : 0 is kontakt, 1 is geen kontakt. In plaats van een joystick kunnen ook maximaal zes "paddles" aangesloten worden.

Hoewel de meeste MSX-machines twee aansluitingen voor een joystick hebben, kan er maar één tegelijk ingelezen worden. Welke van beide dat zal zijn, wordt bepaald door bit 6 in register 15.

Bit 6, het "keyboard mode" bit wordt op Europese machines niet gebruikt. Op Japanse machines heeft het te maken met een verbinding in de computer die de karakterset bepaalt.

Bit 7, het "cassette input" bit, leest het signaal dat van de EAR output van de cassette komt. Dit signaal wordt eerst door een comparator-schakeling vrijgemaakt van onzuiverheden en op digitaal niveau gebracht. Verder wordt dit signaal niet bewerkt.

Register 15

7	6	5	4	3	2	1	0
Kana LED	Joy. Sel.	Puls 2	Puls 1	1	1	1	1

Fig. 33

Dit register dient om naar poort B te schrjven. De laagste vier bits zijn via TTL (open collector)-buffers verbonden met de aansluitpennen 6 en 7 van elke joystick-aansluiting. Normaal zijn die bits 1, wanneer een paddle of een joystick aangesloten is, zodat de aansluitpennen als inputs kunnen werken. Wanneer een "touchpad" aangesloten wordt, dienen ze als output met een handshake-protocol.

De twee puls-bits wekken een korte positieve puls op naar alle paddles die aan connector 1 of 2 aangesloten zijn. Elke paddle bevat een monostabiele timer met een regelbare weerstand, die de lengte van zijn puls bepaalt. Is de timer aangezet, dan kan de stand van de regelbare weerstand berekend worden door te tellen hoelang het duurt vooraleer de monostabiele timer in zijn ruststand terugkeert.

Bit 6, het joystick select bit, legt vast welke joystick met poort A verbonden is voor input : als het bit 0 is, werkt joystick 1; is het bit 1, dan werkt joystick 2.

Bit 7, het Kana Led bit, wordt op Europese machines niet gebruikt. Op Japanse machines stuurt deze bit een indicator, die aangeeft in welke mode het toetsenbord werkt.

HOOFDSTUK 4

Het ROM BIOS

Wanneer iemand efficiënte machinetaalprogramma's wil schrijven die betrouwbaar werken, is het belangrijk dat hij of zij weet hoe de MSX ROM in elkaar zit.

Bijna elk programma, met inbegrip van de BASIC Interpreter zelf, heeft een aantal fundamentele functies nodig om te kunnen werken, zoals de sturing van het scherm en een printer, de decodering van een toetsenbord, en andere functies die met de hardware verband houden. Door deze routines van de BASIC Interpreter gescheiden te houden, zijn ze voor elk toepassingsprogramma toegankelijk. Het deel van de ROM tussen 0000h en 268Bh bestaat hoofdzakelijk uit dergelijke routines. Dit deel heet : het ROM BIOS, het BASIC Input Output System.

Dit hoofdstuk bevat een functionele beschrijving van elke routine in het ROM BIOS die als op zichzelf staand herkend kan worden. Er wordt met speciale aandacht stilgestaan bij de "standaard"-routines. Deze routines werden door Microsoft gedocumenteerd, en blijven gegarandeerd gelijk werken, ook al wordt de software of hardware van de computer gewijzigd.

De eerste paar honderd bytes van de ROM bevatten Z80 JP-instructies. Door die organisatie krijgen de hierna beschreven routines vaste "aanspreekadressen". Om nog te kunnen werken met eventuele latere versies van de software van MSX-computers, is het aan te raden dat toepassingsprogramma's enkel met deze jump-tabel werken, en niet rechtstreeks met de startadressen van de routines zelf.

De beschrijving van de ROM begint bij deze lijst van toegangs-adressen tot de standaard-routines. Bij de toegangspunten zelf staat een kort commentaar; de volledige beschrijving volgt bij de afzonderlijke routines.

Data-gebieden

Waarschijnlijk zullen de meeste gebruikers de ROM tot op bepaalde hoogte willen disassembleren (de volledige listing beslaat bijna vierhonderd bladzijden). Om deze taak wat te verlichten, geven we hieronder de data-gebieden, dat zijn gebieden waarin geen werkende Z80-code staat.

0004h-0007h	185Dh-1863h	4B3Ah-4B4Ch	73E4h-73E4h
002Bh-002Fh	1B97h-1BAAh	4C2Fh-4C3Fh	752Eh-7585h
0508h-050Dh	1BBFh-23BEh	555Ah-5569h	7754h-7757h
092Fh-097Fh	2439h-2459h	5D83h-5DB0h	7BA3h-7BCAh
0DA5h-0EC4h	2CF1h-2E70h	6F76h-6F8Eh	7ED8h-7F26h
1033h-105Ah	3030h-3039h	70FFh-710Ch	7F41h-7FB6h
1061h-10C1h	3710h-3719h	7182h-7195h	7FBEh-7FFFh
1233h-1252h	392Eh-3FE1h	71A2h-71B5h	
13A9h-1448h	43B5h-43C3h	71C7h-71DAh	
160Bh-1612h	46E6h-46E7h	72A6h-72B9h	

Deze datagebieden gelden voor de Engelse ROM. In de Japanse ROM zijn kleine verschillen, die betrekking hebben op de decodering van het toetsenbord en de karakterset. De ROMs verschillen enkel van elkaar wat deze gebieden betreft; het grote deel van de code is voor beide gevallen identiek.

Terminologie

In dit hoofdstuk wordt dikwijls verwezen naar standaard-routines en naar de variabelen in het Werkgebied (Workspace). Bij die verwijzingen gebruiken we de naam die Microsoft aanbeveelt, in hoofdletters, bijvoorbeeld : "de standaardroutine FILVRM" of "SCRMOD wordt geset". Naar subroutines zonder naam wordt verwezen middels een adres tussen haakjes : "het scherm wordt gewist (0777h)". Wanneer de Z80 statusvlaggen worden vermeld, gebruiken we conventionele aanduidingen uit de assembler-taal. De uitdrukking "vlag C" betekent bijvoorbeeld dat de Carryvlag geset wordt; "vlag NZ" wil zeggen dat de Zerovlag gereset wordt. De termen "EI" en "DI" duiden respectievelijk aan dat de interrupts in - of buiten werking worden gesteld.

OVERZICHT VAN DE JUMP-TABEL

Adres	naam	naar	functie
0000H	CHKRAM	02D7H	kontroleer RAM bij aanzetten
0004H	-	-	2 byte-adres van karakterset in ROM
0006H	-	-	nummer van VDP data-poort
0007H	-	-	nummer van VDP data-poort
0008H	SYNCHR	2683H	kontroleer karakter in BASIC programma
000BH	-	-	NOP
000CH	RDSLT	01B6H	lees RAM in een willekeurig slot
000FH	-	-	NOP
0010H	CHRGTR	2686H	haal volgende BASIC-karakter
0013H	-	-	NOP
0014H	WRSLT	01D1H	Schrijf naar RAM in willekeurig slot
0017H	-	-	NOP
0018H	OUTDO	1B45H	stuur output naar huidig kanaal
001BH	-	-	NOP
001CH	CALSLT	0217H	CALL een routine in willekeurig slot
001FH	-	-	NOP
0020H	DCOMPR	146AH	vergelijk registerparen HL en DE
0023H	-	-	NOP
0024H	ENASLT	025EH	maak een bepaald slot permanent actief
0027H	-	-	NOP
0028H	GETYPR	2689H	zoek type van BASIC operand
002BH	-	-	vijf bytes : nr. van de versie
0030H	CALLF	0205H	CALL een routine in een bepaald slot
0033H	-	-	5 maal NOP
0038H	KEYINT	0C3CH	interrupt-routine, aftasting toetsenbord
003BH	INITIO	049DH	initialiseer I/O hardware
003EH	INIFNK	139DH	initialiseer functietoetsen
0041H	DISSCR	0577H	maak scherm inactief
0044H	ENASCR	0570H	maak scherm actief
0047H	WRTVDP	057FH	schrijf naar een VDP register
004AH	RDVRM	07D7H	lees een byte uit VRAM
004DH	WRTVRM	07CDH	schrijf een byte naar VRAM
0050H	SETRD	07ECH	maak VDP klaar voor lees-operatie
0053H	SETWRT	07DFH	maak VDP klaar voor schrijf-operatie
0056H	FILVRM	0815H	vul een VRAM-blok met een bepaalde waarde
0059H	LDIRMV	070FH	kopieer blok VRAM naar geheugen
005CH	LDIRVM	0744H	kopieer blok geheugen naar VRAM
005FH	CHGMOD	084FH	verander de VDP-mode
0062H	CHGCLR	07F7H	verander de VDP-kleuren
0065H	-	-	NOP
0066H	NMI	1398H	Non-maskable interrupt-routines
0069H	CLRSPR	06A8H	wis alle sprites
006CH	INITXT	050EH	initialiseer VDP voor 40 x 24 tekstmode
006FH	INIT32	0538H	initialiseer VDP voor 32 x 24 tekstmode
0072H	INIGRP	05D2H	initialiseer VDP voor grafische mode
0075H	INIMLT	061FH	initialiseer VDP voor veelkleuren-mode
0078H	SETTXT	0594H	stel VDP in op 40 x 24 tekstmode
007BH	SET32	05B4H	stel VDP in op 32 x 24 tekstmode
007EH	SETGRP	0602H	stel VDP in op grafische mode
0081H	SETMLT	0659H	stel VDP in op veelkleurenmode
0084H	CALPAT	06E4H	bereken adres van sprite-patroon
0087H	CALATR	06F9H	bereken adres van sprite-attributen
008AH	GSPSIZ	0704H	zoek grootte van sprite
008DH	GRPPRT	1510H	print teken op grafisch scherm

0090H	GICINI	04BDH	initialiseer de PSG (van General Instr.)
0093H	WRTPSG	1102H	schrijf naar een PSG-register
0096H	RDPSG	110EH	lees uit een PSG-register
0099H	STRTMS	11C4H	begin muziek uit te lezen
009CH	CHSNS	0D6AH	tast toetsenbord-buffer af voor karakter
009FH	CHGET	10CBH	haal karakter uit buffer (wacht er op)
00A2H	CHPUT	08BCH	output van karakter naar scherm
00A5H	LPTOUT	085DH	output van karakter naar printer
00A8H	LPTSTT	0884H	test of printer klaar is
00ABH	CNVCHR	089DH	zet karakter met grafische header byte om
00AEH	PINLIN	23BFH	lees een ingetypte regel (editor)
00B1H	INLIN	23D5H	lees een ingetypte regel (editor)
00B4H	QINLIN	23CCH	zet een "?" en lees een ingetypte regel
00B7H	BREAKX	046FH	direkte controle of CRTL-STOP ingedrukt
00BAH	ISCNTC	03FBH	controleer CTRL-STOP toetsen
00BDH	CKCNTC	10F9H	controleer CTRL-STOP toetsen
00C0H	BEEP	1113H	doe "beep"
00C3H	CLS	0848H	wis het scherm
00C6H	POSIT	088EH	vul de cursorpositie in
00C9H	FNKSB	0B26H	is tekst functietoetsen zichtbaar ?
00CCH	ERAFNK	0B15H	tekst functietoetsen van scherm halen
00CFH	DSPFNK	0B2BH	tekst functietoetsen op scherm zetten
00D2H	TOTEXT	083BH	keer terug naar tekst-mode
00D5H	GTSTCK	11EEH	lees joystick-toestand
00D8H	GTTRIG	1253H	lees vuurknop-toestand
00DBH	GTPAD	12ACH	lees touchpad-toestand
00DEH	GTPDL	1273H	lees paddle-toestand
00E1H	TAPION	1A63H	zet cassette input aan
00E4H	TAPIN	1ABCH	lees cassettesignaal
00E7H	TAPIOF	19E9H	zet cassette input uit
00EAH	TAPOON	19F1H	zet cassette output aan
00EDH	TAPOUT	1A19H	stuur signaal naar cassette
00F0H	TAPOOF	19DDH	zet cassette output uit
00F3H	STMOTR	1384H	zet motor aan of uit
00F6H	LFTQ	14EBH	ruimte over in muziek-string
00F9H	PUTQ	1492H	zet een byte in de muziek-string
00FCH	RIGHTC	16C5H	schuif pixel 1 positie naar rechts
00FFH	LEFTC	16EEH	schuif pixel 1 positie naar links
0102H	UPC	175DH	schuif pixel 1 positie omhoog
0105H	TUPC	173CH	test of UPC kan en zo ja, doe het
0108H	DOWNC	172AH	schuif pixel 1 positie omlaag
010BH	TDOWNC	170AH	test of DOWNC kan en zo ja, doe het
010EH	SCALXY	1599H	breng grafische coördinaten op schaal
0111H	MAPXYC	15DFH	bereken adres van grafische coördinaten
0114H	FETCHC	1639H	bereken adres van een pixel
0117H	STOREC	1640H	zet adres van een pixel weg
011AH	SETATR	1676H	vul attribute-byte in
011DH	READC	1647H	lees het attribute van een pixel
0120H	SETC	167EH	vul attribute-byte van pixel in
0123H	NSETCX	1809H	vul attribute-bytes van aantal pixels in
0126H	GTASPC	18C7H	bereken verhouding bij cirkel
0129H	PNTINI	18CFH	initialiseer PAINT-routine
012CH	SCANR	18E4H	zoek pixel naar rechts
012FH	SCANL	197AH	zoek pixel naar links
0132H	CHGCAP	0F3DH	wijzig toestand van de CAPS-LED
0135H	CHGSND	0F7AH	wijzig de klik bij toetsindruk
0138H	RSLREG	144CH	lees het primair slotregister
013BH	WSLREG	144FH	schrijf naar het primair slotregister
013EH	RDVDP	1449H	lees het VDP-statusregister
0141H	SNSMAT	1452H	lees een rij van de toetsenbord-matrix

0144H	PHYDIO	148AH	voor diskdrive, geen effect
0147H	FORMAT	148EH	voor diskdrive, geen effect
014AH	ISFLIO	145FH	in/uit via file-buffer ?
014DH	OUTDLP	1B63H	geformatteerde output naar printer
0150H	GETVCP	1470H	haal pointer van muziekkanaal
0153H	GETVC2	1474H	haal pointer van muziekkanaal
0156H	KILBUF	0468H	wis de toetsenbord-buffer schoon
0159H	CALBAS	01FFH	CALL naar BASIC vanuit elk slot
015CH	-	-	NOPs tot 01B5H,voorzien voor uitbreiding

Op de volgende bladzijden zullen we de verschillende routines één voor één bespreken in de volgorde van hun startadres, dus niet van hun toegangsadres in de jump-tabel.

adres   01B6H
----
naam   RDSLT
in   A=slot-identificatie, HL=adres
uit   A= gelezen byte
wijzigt AF,BC,DE, DI

Standaard-routine om een byte in het geheugen te lezen, in elk slot. De slot-identificatie bestaat uit het nummer van het primair slot, het nummer van het secundair slot, en een vlag.

7	6	5	4	3 2	1 0
vlag	0	0	0	SSLOT#	PSLOT#

Fig. 34 : slot-identificatie

De vlag is normaal 0, maar moet 1 zijn indien in de identificator een secundair slot-nummer vervat zit. Eerst worden het adres en de identificator verwerkt (027EH) tot een aantal bitmaskers, die met het betrokken slotregister vergeleken zullen worden. Werd een secundair slot gespecifieerd, dan wordt eerst het secundaire slotregister zo gewijzigd dat de geëigende bladzijde van dat slot geselecteerd wordt (02A3H). Dan wordt het primaire slot in het geheugenbereik van de Z80 geschakeld, het byte wordt gelezen en het primair slot hersteld in zijn oorspronkelijke staat via de RDPRIM-routine in het Werkgebied. Tenslotte wordt nog, indien een secundair slotnummer in de identificator was opgenomen, het secundair slotregister in zijn oorspronkelijke staat hersteld (01ECH).

Let er wel op dat, tenzij het om het slot gaat dat het Werkgebied bevat, elke poging om uit geheugenbladzijde 3 te lezen (C000H tot FFFFH) een crash veroorzaakt, doordat RDPRIM zichzelf buiten het bereik van de Z80 schakelt. Alle routines die geheugen in- of uitschakelen stellen ook de interrupts buiten werking.

adres   01D1H
----
naam   WRSLT
in   A=slot identif., HL=adres, E=te schrijven byte
uit   n.v.t.
wijzigt AF,BC,D, DI

Dit is de standaard-routine om een byte in het geheugen te schrijven, in een willekeurig slot. De werking ervan is fundamenteel dezelfde als van de RDSLT-routine, behalve dat in het Werkgebied de WRPRIM-routine in plaats van RDPRIM gebruikt wordt.

```
adres     01FFH
--------------
naam      CALBAS
in        IX-adres
uit       n.v.t.
wijzigt   AF',BC',DE',HL',IY, DI
```

Deze standaard-routine roept een subroutine aan in de BASIC Interpreter, vanuit elk slot. Dit zal vrijwel altijd gebeuren vanuit een machinetaal-programma dat in een uitbreidingsROM op bladzijde 1 (4000H tot 7FFFH) loopt. Het I registerdeel van IY wordt geladen met de identificator van het MSX-ROMslot (00H), en de routine gaat verder naar CALSLT.

```
adres     0205H
--------------
naam      CALLF
in        n.v.t.
uit       n.v.t.
wijzigt   AF',BC',DE',HL',IX,IY, DI
```

Via deze standaardroutine kan een CALL naar elk adres in elk slot gebeuren. Slot-identificatie en het CALL-adres worden als parameters gegeven en niet via registers, om als hook gebruikt te kunnen worden (zie hoofdstuk 6). Een voorbeeld :

```
          RST 30H
          DEFB (slot ident.)
          DEFW (adres)
          RET
```

Eerst wordt de slot-identificatie opgehaald, en in het MSB van IY gezet. Vervolgens wordt het adres in IX geladen, waarna de controle overgegeven wordt aan CALSLT.

```
adres     0217H
--------------
naam      CALSLT
in        IY (MSB)-slot-identif., IX-adres
uit       n.v.t.
wijzigt   AF',BC',DE',HL', DI
```

Dit is de eigenlijke algemene standaardroutine waarmee een CALL kan plaatsvinden naar elk slot. Ze werkt in de grond op dezelfde manier als RDSLT, behalve dat nu CLPRIM in het Werkgebied wordt gebruikt, in plaats van RDPRIM. CALBAS en CALLF zijn enkel speciale toegangs-adressen tot deze routine, die maken dat de programmeur zelf minder code moet schrijven.

```
adres     025EH
--------------
naam      ENASLT
in        A-slot-identif., HL-adres
uit       n.v.t.
wijzigt   AF,BC,DE, DI
```

Deze standaardroutine schakelt een bladzijde vanuit elk slot permanent in het geheugengebied van de processor. In tegenstelling tot de RDSLT, WRSLT en CALSLT-routines gebeurt

hier het inschakelen van het primaire slot rechtstreeks, en niet via een routine in het Werkgebied. Dit houdt in dat het opgeven van een adres op bladzijde 0 (0000H tot 3FFFH) voor een ogenblikkelijke crash zorgt.

adres 027EH

Op dit adres start een routine die gebruikt wordt door alle routines die geheugen schakelen. De routine bewerkt een adres (in HL) en een slot-identificator (in A) tot een groep bitmaskers. Een slot-identificator "FxxxSSPP" en een adres op bladzijde 1 (tussen 4000H en 7FFFH) zou bijvoorbeeld het volgende resultaat opleveren :

```
register B : 00 00 PP 00 (OR-masker)
register C : 11 11 00 11 (AND-masker)
register D : PP PP PP PP (gedupliceerd)
register E : 00 00 11 00 (bladzijde-masker)
```

De B en C registers worden afgeleid van het nummer van het primaire slot en het bladzijde-masker. Later worden deze registers gebruikt om het nieuwe primaire slot-nummer in te voegen in de bestaande inhoud van het primair slot-register. Register D bevat vier keer het primair slot-nummer, en het E-register bevat het bladzijde-masker. Dit masker wordt gemaakt aan de hand van de hoogste twee bits van het adres (om de bladzijde te bepalen). Dan worden de bits naar de passende plaats verschoven. Deze registers worden later gebruikt tijdens het omschakelen van de secundaire slots.

Op het einde van deze routine wordt bit 7 van de slot-identificator getest, om te zien of er een secundair slot werd gespecificeerd. Was dit zo, dan wordt de M vlag geset.

adres 02A3H

Deze routine wordt gebruikt door de standaard-routines die geheugen omschakelen, om de inhoud van een secundair slot-register te wijzigen. De slot-identificatie wordt in A verschaft; registers D en E bevatten de bitmaskers, zoals in de vorige routine.

Bits 6 en 7 van D worden eerst naar het primair slotregister gecopieerd. Daardoor wordt bladzijde 3 ingeschakeld van het primair slot dat door de slot-identificator werd bepaald, en wordt het secundair slotregister aanspreekbaar. Dit wordt dan gelezen (adres FFFFH) en met het geïnverteerde bladzijde-masker worden de gewenste twee bits vrijgemaakt. Het secundair slot-nummer wordt tot op de goede plaats geschoven en ermee gecombineerd. De verkregen combinatie wordt in het secundair slotregister geschreven, en het primair slotregister wordt in zijn oorspronkelijke stand teruggezet.

adres 02D7H
----

naam CHKRAM
in n.v.t.
uit n.v.t.
wijzigt AF,BC,DE,HL,SP

Deze standaardroutine initialiseert het geheugen bij het aanzetten van de machine. Ze controleert op een niet-destructieve manier of er RAM aanwezig is op bladzijden 2 en 3 van de zestien mogelijke slots. Daarna beschrijft ze de primaire en secundaire slotregisters zodanig dat het grootste gevonden gebied in het geheugenbereik van de processor geschakeld wordt. Het hele Werkgebied wordt met nullen gevuld (van F380H tot FFC9H). In EXPTBL en SLTTBL wordt ingevuld welke uitbreidings-interfaces aanwezig zijn. Interrupt Mode 1 wordt ingesteld, en de routine springt naar de rest van de initialisatieroutine op 7C76H.

adres 03FBH

---

naam ISCNTC
in n.v.t.
uit n.v.t.
wijzigt AF, EI

Deze standaardroutine controleert of de "CTRL/STOP" of de "STOP"-toetsen werden ingedrukt. De BASIC Interpreter gebruikt ze op het einde van elk statement, om te zien of het programma beëindigd dient te worden. Vooraf wordt de inhoud van BASROM gelezen. Verschilt die van nul, dan wordt de routine afgebroken. Dit heeft tot doel, te voorkomen dat gebruikers inbreken in uitbreidings-ROMs die BASIC programma's bevatten. Verder wordt INTFLG continu uitgelezen, om te zien of de interrupt-routines de codes van "CTRL/STOP" of "STOP" daar hebben gezet (codes 03H of 04H). Werd "STOP" ingedrukt, dan wordt de cursor zichtbaar gemaakt (09DAH) en wordt INTFLG voortdurend bekeken tot een van beide codes weer voorkomt. Dan wordt de cursor van het scherm gehaald (02A7H) en als de "STOP"-toets werd ingedrukt stopt de routine.

Werd een "CTRL/STOP"-code ontdekt, dan wordt de toetsenbord-buffer leeggemaakt via de KILBUF-routine. In TRPTBL wordt nagegaan of een "ON STOP GOSUB"-statement actief is. Is dat het geval, dan wordt het desbetreffende deel van TRPTBL aangepast (0EF1H) en de routine stopt : de "GOSUB" wordt verder door de Interpreter afgehandeld. Werd "CTRL/STOP" ontdekt, dan zorgt de ENASLT-routine ervoor dat bladzijde 1 van de MSX-ROM ingeschakeld wordt - voor het geval de routine door een uitbreidingsROM wordt gebruikt - en de controle wordt overgedragen aan de "STOP"-statement routine (63E6H).

adres 0468H

---

naam KILBUF
in n.v.t.
uit n.v.t.
wijzigt HL

Deze standaardroutine maakt de toetsenbord-buffer leeg. Die buffer (KEYBUF) kan 40 karakters bevatten, die op voorhand ingetypt kunnen worden en één na één verwerkt worden. Bij die buffer horen twee pointers : PUTPNT en GETPNT. PUTPNT duidt aan waar de interruptroutines een ingetoetst karakter moeten zetten, terwijl GETPNT aangeeft op welk adres in de buffer, toepassingsprogramma's karakters kunnen uitlezen. Het aantal karakters in de buffer wordt aangegeven door het verschil tussen

deze twee pointers. KEYBUF wordt leeggemaakt door simpelweg deze twee pointers gelijk te maken.

```
adres    046FH
--------------
naam     BREAKX
in       n.v.t.
uit      vlag C indien "CTRL/STOP" ingedrukt werd
wijzigt  AF
```

Deze standaardroutine peilt rechtstreeks rij 6 en 7 van het toetsenbord, om te zien of "CTRL" en "STOP" tegelijk worden ingedrukt. Is dat zo, dan wordt KEYBUF leeggemaakt en rij 7 van OLDKEY aangepast, dit laatste om te voorkomen dat de interrupt- routines de toetsen zouden detecteren. Voor toepassings- programma's is deze routine vaak beter te gebruiken dan ISCNTC, omdat ze ook werkt wanneer de interrupts niet worden ontvangen (bijvoorbeeld tijdens input of output naar cassette) en de controle rechtstreeks aan het programma wordt teruggegeven, en niet via de Interpreter.

```
adres    049DH
--------------
naam     INITIO
in       n.v.t.
uit      n.v.t.
wijzigt  AF,E, EI
```

Deze standaardroutine initialiseert de PSG en de Centronics status-poort. Register 7 van de PSG wordt op 80H gezet, waardoor poort B van de PSG voor output en poort A voor input zal gebruikt worden. Register 15 van de PSG wordt met CFH geladen, om de hardware die de joystick-aansluiting stuurt, te initialiseren. Register 14 van de PSG wordt uitgelezen en het Keyboard Mode bit wordt in KANAMD gezet. Dit is van geen belang voor Europese machines.

Tenslotte wordt FFH op de Centronics statuspoort gezet (In/Out-poort 90H), waardoor het strobe-signaal hoog wordt. Dan gaat de routine naar GICINI, waar de initialisatie afgewerkt wordt.

```
adres    04BDH
--------------
naam     GICINI
in       n.v.t.
uit      n.v.t.
wijzigt  EI
```

Deze standaardroutine initialiseert de PSG en de variabelen in het Werkgebied die te maken hebben met het "PLAY"-statement. QUETAB, VCBA, VCBB, en VCBC worden eerst op de waarden gezet die in hoofdstuk 6 gegeven worden. De registers 8, 9 en 10 van de PSG worden op amplitude 0 gezet, en register 7 op B8H. Daardoor wordt de toongenerator op elk kanaal aangesloten, en de ruisgenerator van elk kanaal afgesloten.

adres 0508H TABEL

Deze tabel van zes bytes bevat de parameters voor het "PLAY"-statement zoals die oorspronkelijk in VCBA, VCBB en VCBC werden gezet door de standaardroutine GICINI : octaaf 4, lengte 4, tempo 120, volume 88H, envelope 00FFH.

adres 050EH

---

naam INITXT
in n.v.t.
uit n.v.t.
wijzigt AF,BC,DE,HL,EI

Door deze standaard routine wordt de VDP geïnitialiseerd voor de 40 x 24 tekstmode. Het scherm wordt tijdelijk buiten werking gesteld via de standaardroutine DISSCR; SCRMOD en OLDSCR worden op 0 gezet. De parameters, nodig voor de standaardroutine CHPUT worden ingevuld door LINL40 naar LINLEN, TXTNAM naar NAMBAS en TXTCGP naar CGPBAS te copiëren. De kleuren voor de VDP worden vervolgens ingevuld door de standaardroutine CHGCLR en het scherm wordt gewist (077EH). De karakterset in gebruik wordt gecopieerd naar de Karakterpatroon-tabel in VRAM (071EH). Tenslotte worden de VDP-mode en de basis-adressen vastgelegd door de standaardroutine SETTXT, en het scherm wordt opnieuw in werking gesteld.

adres 0538H

---

naam INIT32
in n.v.t.
uit n.v.t.
wijzigt AF,BC,DE,HL,EI

Door deze standaardroutine wordt de VDP geïnitialiseerd voor de 32 x 24 tekstmode. Het scherm wordt tijdelijk buiten werking gesteld via de standaardroutine DISSCR en SCRMOD en OLDSCR worden op 01H gezet. De parameters, nodig voor de standaardroutine CHPUT worden ingevuld door LINL32 naar LINLEN, T32NAM naar NAMBAS, T32CGP naar CGPBAS, T32PAT naar PATBAS en T32ATR naar ATRBAS te copiëren. De kleuren voor de VDP worden vervolgens ingevuld door de standaardroutine CHGCLR en het scherm wordt gewist (077EH). De karakterset in gebruik wordt gecopieerd naar de Karaktertabel in VRAM (071EH) en alle sprites worden gewist. (06BBH). Tenslotte worden de VDP-mode en de basisadressen vastgelegd door de standaardroutine SETT32, en het scherm wordt opnieuw actief gemaakt.

adres 0570H

---

naam ENASCR
in n.v.t.
uit n.v.t.
wijzigt AF,BC,EI

Deze standaardroutine maakt het scherm actief, door bit 6 van het VDP-moderegister 1 te setten.

adres 0577H
----
naam DISSCR
in n.v.t.
uit n.v.t.
wijzigt AF,BC,EI

Deze standaardroutine maakt het scherm inactief, door bit 6 van het VDP moderegister 1 te resetten.

adres 057FH
----
naam WRTVDP
in B=data byte, C=nummer van VDP moderegister
uit n.v.t.
wijzigt AF,B,EI

Deze standaardroutine wordt gebruikt om een byte in een van de VDP-moderegisters te zetten. Eerst wordt het registernummer naar de commandopoort van de VDP geschreven, en dan het databyte. Dit wordt daarna gecopieerd naar het Werkgebied, RG0SAV tot RG7SAV, waar de inhoud van de registers bijgehouden wordt.

adres 0594H
----
naam SETTXT
in n.v.t.
uit n.v.t.
wijzigt AF,BC,DE,HL,EI

Met deze standaardroutine wordt de VDP gedeeltelijk klaargemaakt voor de 40 x 24 tekstmode. De Modebits M1, M2 en M3 in de VDP-registers 0 en 1 worden in de goede stand gezet. Daarna worden de vijf basisadressen van de tabellen in VRAM, te beginnen met TXTNAM, van het Werkgebied gecopieerd naar de VDP moderegisters 2, 3, 4, 5 en 6 (0677H).

adres 05B4H
----
naam SETT32
in n.v.t.
uit n.v.t.
wijzigt AF,BC,DE,HL,EI

Met deze standaardroutine wordt de VDP gedeeltelijk klaargemaakt voor de 32 x 24 tekstmode. De Modebits M1, M2 en M3 in de VDP-registers 0 en 1 worden in de goede stand gezet. Daarna worden de vijf basisadressen van de tabellen in VRAM, te beginnen met T32NAM, van het werkgebied gecopieerd naar de VDP moderegisters 2, 3, 4, 5 en 6 (0677H).

adres 05D2H
----
naam INIGRP
in n.v.t.
uit n.v.t.
wijzigt AF,BC,DE,HL,EI

Deze standaardroutine initialiseert de VDP voor de grafische mode. Het scherm wordt tijdelijk buiten werking gesteld via de standaardroutine DISSCR en in SCRMOD wordt 2 gezet. De parameters, nodig voor de standaardroutine GRPPRT, worden klaargezet door het copiëren van GRPPAT naar PATBAS en van GRPATR naar ATRBAS. Het pilootpatroon voor de karaktercodes wordt dan gecopieerd naar de Namentabel van de VDP. Het scherm wordt schoongemaakt (07A1H) en alle sprites gewist (06BBH). Tenslotte wordt de VDP mode ingesteld en worden de basisadressen aangepast via de standaardroutine SETGRP, en het scherm wordt opnieuw actief gemaakt.

adres   0602H
----
naam   SETGRP
in   n.v.t.
uit   n.v.t.
wijzigt AF,BC,DE,HL,EI

Door deze standaardroutine wordt de VDP gedeeltelijk geïnitialiseerd voor de grafische mode. Modebits M1, M2 en M3 in de VDP moderegisters 0 en 1 worden in de goede stand gezet. De vijf basisadressen van de VRAM-tabellen worden dan vanuit het Werkgebied naar de VDP - moderegisters 2, 3, 4, 5 en 6 gecopieerd (0677H).

adres   061FH
----
naam   INIMLT
in   n.v.t.
uit   n.v.t.
wijzigt AF,BC,DE,HL,EI

Door deze standaardroutine wordt de VDP klaargemaakt voor de veelkleurenmode. Het scherm wordt tijdelijk buiten werking gesteld via de standaardroutine DISSCR en in SCRMOD wordt 3 gezet. De parameters, nodig voor de standaardroutine GRPPRT worden klaargezet door het copiëren van MLTPAT naar PATBAS en van MLTATR naar ATRBAS. Het pilootpatroon voor de karaktercodes wordt gecopieerd naar de Namentabel van de VDP, het scherm wordt schoongemaakt (07B9H) en alle sprites gewist (06BBH). Tenslotte worden de VDP mode en basisadressen ingevuld via de standaardroutine SETMLT, en het scherm wordt weer actief gemaakt.

adres   0659H
----
naam   SETMLT
in   n.v.t.
uit   n.v.t.
wijzigt AF,BC,DE,HL,EI

Deze standaardroutine maakt de VDP gedeeltelijk klaar voor de veelkleurenmode. Modebits M1, M2 en M3 in de VDP moderegisters 0 en 1 worden in de goede stand gezet. De vijf basisadressen van de VRAM-tabellen, te beginnen met MLTNAM, worden dan vanuit het Werkgebied naar de VDP moderegisters 2, 3, 4, 5 en 6 gecopieerd.

adres   0677H

Deze subroutine wordt gebruikt door de standaardroutine SETTXT, SETT32, SETGRP en SETMLT, om een blok van vijf basisadressen voor tabellen te copiëren vanuit het Werkgebied naar de VDP registers 2, 3, 4, 5 en 6. Wanneer de routine wordt aangeroepen, bevat HL het adres van de groep van vijf adressen. Eën na één worden de basis- adressen opgehaald, het nodige aantal plaatsen verschoven, en dan in het juiste moderegister geplaatst door de standaardroutine WRTVDP.

adres   06A8H
----

naam   CLRSPR
in   n.v.t.
uit   n.v.t.
wijzigt   AF,BC,DE,HL,EI

Deze standaardroutine wist alle sprites. De patroontabel van de sprites wordt eerst over de hele lengte van 2 K met nullen gevuld via de standaardroutine FILVRM. Het vertikale coördinaat van de tweeëndertig attributen-blokken van de sprites wordt vervolgens op -47 (D1H) gezet, waardoor de sprite boven de rand van het scherm staat. Het horizontale coördinaat wordt niet gewijzigd.

De patroonnummers in de attributentabel worden geïnitialiseerd met de serie 0,1,2,3,4,...,31 voor 8x8 sprites, of met de serie 0,4,8,12,16,...,124 voor 16x16 sprites. Welke serie gebruikt wordt, bepaalt het grootte-bit in VDP moderegister 1. Tenslotte wordt het kleurbyte van elk attributenblok ingevuld met de kleur die in FORCLR zit; oorspronkelijk is dit wit.

Deze routine heeft geen invloed op de grootte- en vergrotings-bits in VDP moderegister 1. Doordat de standaardroutine INIT32, INIGRP en INIMLT deze routine gebruiken vanaf adres 06BBH, veranderen ze geen van drieën iets aan de patronentabel.

adres   06E4H
----

naam   CALPAT
in   A = nummer van sprite-patroon
uit   HL = adres van dat sprite-patroon
wijzigt   AF,DE,HL

Met deze standaardroutine wordt het adres van een sprite-patroon berekend. Eerst wordt het patroon-nummer met acht vermenigvuldigd. Zijn er 16 x 16 sprites geselecteerd, dan wordt dit resultaat nogmaals met vier vermenigvuldigd. Dit resultaat wordt opgeteld bij het basisadres van de patronentabel, dat in PATBAS staat, en dit geeft het gevraagde adres.

Dit systeem van nummering klopt met het systeem dat de BASIC Interpreter gebruikt, maar niet met dat van de VDP wanneer 16x16 sprites gekozen werden. Terwijl de Interpreter bijvoorbeeld het tweede patroon als nummer 1 benoemt, is het eigenlijk patroon nummer 4 in de VDP. Dit houdt in dat, wanneer het om 16x16 sprites gaat, deze routine als hoogste patroon-nummer 63 zou

mogen toelaten. In feite wordt dit niet gecontroleerd. Hoge patroon-nummers zullen dus een adres opleveren boven 3FFFH. Wanneer dergelijke hoge adressen door de andere VDP-routines overgenomen worden, zullen die terug in het lagere bereik, vanaf 0000H terechtkomen, en de tabel met karakterpatronen in VRAM in de war brengen.

adres 06F9H
----

naam CALATR
in A - nummer van de sprite
uit HL - adres van attribuut van die sprite
wijzigt AF,DE,HL

Deze standaardroutine berekent het adres van het attributenblok van een sprite. Het sprite-nummer, tusen 0 en 31, wordt vermenigvuldigd met vier en dit resultaat opgeteld bij het basisadres van de attributentabel, dat in ATRBAS staat.

adres 0704H
----

naam GSPSIZ
in n.v.t.
uit A - aantal bytes in sprite-patroon (8 of 32)
wijzigt AF

Deze standaardroutine levert het aantal bytes op, dat elk sprite-patroon in de patronentabel inneemt. Dit wordt bereikt door het grootte-bit in VDP-moderegister 1 te onderzoeken.

adres 070FH
----

naam LDIRMV
in BC-lengte,DE-RAM-adres,HL-VRAM-adres
uit n.v.t.
wijzigt AF,BC,DE,EI

Deze standaardroutine copieert een geheugenblok vanuit VRAM naar het RAM-geheugen. Het startadres in VRAM wordt doorgegeven via de standaardroutine SETRD en vervolgens worden de opeenvolgende bytes uitgelezen van de VDP Datapoort, en in het geheugen gezet.

adres 071EH

Met deze routine kan een karakterset van 2 K lang in elke mode naar de VDP Karaktertabel gecopieerd worden. Het basisadres van de tabel in VRAM wordt uit CGPBAS gehaald. Het startadres van de karakterset wordt uit CGPNT ingelezen. Om de karakter-data te lezen, wordt de standaardroutine RDSLT gebruikt, zodat de karakterset zich ook in een uitbreidingsROM mag bevinden.

Bij het inschakelen van de machine wordt in CGPNT het adres gezet, dat op adres 0004H staat, dat wil zeggen : adres 1BBFH. CGPNT kan makkelijk gewijzigd worden, waardoor je soms interessante resultaten verkrijgt. Nogal verwarrend werkt bijvoorbeeld : POKE &HF920, &HC7 : SCREEN 0.

adres 0744H

----

naam LDIRMV
in BC = lengte, DE = adres in VRAM, HL = adres in RAM
uit n.v.t.
wijzigt AF,BC,DE,HL,EI

Deze standaardroutine copieert een geheugenblok van RAM naar VRAM. Het startadres in VRAM wordt doorgegeven via de standaardroutine SETWRT, en vervolgens worden de opeenvolgende bytes uit het geheugen naar de VDP datapoort geschreven.

adres 0777H

Deze routine wist het scherm in elke VDP-Mode. In 40x24 en 32x24 tekstmode wordt eerst de Namentabel gevuld met ASCII-spaties. Het basisadres voor die tabel wordt uit NAMBAS gehaald. Dan wordt de cursor linksbovenaan het scherm gezet (0A7FH), en LINTTB (eind van de lijn-tabel) opnieuw geïnitialiseerd. Tenslotte wordt, wanneer dit toegestaan is, de tekst van de funktietoetsen op het scherm gezet door de standaardroutine FNKSB.

In grafische mode wordt eerst de border gekleurd via VDP Moderegister 7 (0832H). Dan wordt de kleurentabel gevuld met de code voor de achtergrondkleur, die uit BAKCLR gehaald wordt, voor alle pixels (0 en 1). Tenslotte wordt de karaktertabel met nullen gevuld.

In veelkleurenmode wordt eerst de border gekleurd via VDP register 7 (0832H), waarna de karaktertabel opgevuld wordt met de achtergrondkleur, die uit BAKCLR gehaald wordt.

adres 07CDH

----

naam WRTVRM
in A = databyte, HL = adres in VRAM
uit n.v.t.
wijzigt EI

Met deze standaardroutine kan een byte naar VRAM geschreven worden. Eerst wordt het VRAM adres doorgegeven via de standaardroutine SETWRT, en dan wordt het databyte naar de Datapoort van de Video Display Processor geschreven. De beide ogenschijnlijk overbodige EX (SP),HL instructies in deze routine - en nog in andere routines - zijn nodig in verband met de timing-eisen van de Video Display Processor.

adres 07D7H

----

naam RDVRM
in HL = adres in VRAM
uit A = (HL)
wijzigt AF, EI

Met deze standaardroutine kan 1 byte uit de VRAM gelezen worden. Eerst wordt het adres doorgegeven via de standaardroutine SETRD, en dan wordt de inhoud ervan uitgelezen via de datapoort van de Video Display Processor.

adres   07DFH
----
naam   SETWRT
in   HL = adres in VRAM
uit   n.v.t.
wijzigt AF, EI

Deze standaardroutine maakt de Video Display Processor klaar om opeenvolgende bytes naar VRAM te schrijven via de datapoort. Het adres in HL wordt naar de commandopoort van de Video Display Processor geschreven in het formaat : LSB eerst, dan MSB (zie fig. 7). Adressen boven 3FFFH tellen door vanaf 0000H, aangezien de hoogste twee bits van het adres niet in aanmerking genomen worden.

adres   07ECH
----
naam   SETRD
in   HL = adres in VRAM
uit   n.v.t.
wijzigt AF, EI

Deze standaardroutine maakt de Video Display Processor klaar om opeenvolgende bytes uit VRAM te lezen via de datapoort. Het adres in registerpaar HL wordt naar de commandopoort van de Video Display Processor geschreven in het formaat : LSB eerst, dan MSB (zie Fig. 7). Adressen boven 3FFFH tellen door vanaf 0000H, aangezien de hoogste twee bits van het adres niet in aanmerking genomen worden.

adres   07F7H
----
naam   CHGCLR
in   n.v.t.
uit   n.v.t.
wijzigt AF,BC,HL,EI

Deze standaard routine stelt de kleuren voor de Video Display Processor in. Eerst wordt in SCRMOD gekeken wat er moet gebeuren. In de 40x24 tekstmode wordt de inhoud van BAKCLR en FORCLR in register 7 van de Video Display Processor gezet, waardoor de kleur van de "1" en "0"-pixels bepaald wordt. Oorspronkelijk zijn die kleuren respectievelijk blauw en wit. In deze mode kan geen aparte borderkleur opgegeven worden; die is dezelfde als de achtergrondkleur. In 32x24 tekstmode, grafische mode of veelkleurenmode wordt de inhoud van BDRCLR in register 7 van de Video Display Processor gezet, waardoor de borderkleur bepaald is. In 32x24 tekstmode wordt de inhoud van BAKCLR en FORCLR ook nog in de hele Kleurentabel gezet, waardoor de kleur van "1" en "0"-pixels bepaald wordt.

adres   0815H
----
naam   FILVRM
in   A = databyte, BC = lengte blok, HL = adres in VRAM
uit   n.v.t.
wijzigt AF,BC,EI

Deze standaard routine vult een VRAM-blok met een bepaalde

waarde (het databyte in register A). Eerst wordt het startadres in VRAM, in registerpaar HL, doorgegeven via de standaardroutine SETWRT. Vervolgens wordt het data byte BC maal naar de datapoort van de Video Display Processor geschreven, om BC opeenvolgende VRAM-adressen te beschrijven.

```
adres     083BH
--------------
naam      TOTEXT
in        n.v.t.
uit       n.v.t.
wijzigt   AF,BC,DE,HL,EI
```

Door deze standaard routine wordt de Video Display Processor teruggezet naar 40x24 of 32x24 tekstmode, wanneer hij op het ogenblik dat de routine aangeroepen wordt, in grafische of veelkleurenmode staat. De routine wordt door de Hoofdlus van de BASIC Interpreter en de "INPUT"-statement-verwerker gebruikt. Bij elke CALL naar de standaardroutine INITXT of INIT32 wordt het mode byte, 0 of 1, in OLDSCR gecopieerd. Wanneer vervolgens naar grafische mode of veelkleurenmode gegaan wordt, en daarna terug naar een van de twee tekstmodes voor input vanaf het toetsenbord, zorgt deze routine ervoor dat de Video Display Processor naar de correcte mode terugkeert.

Vooraf wordt SCRMOD getest. Blijkt dat het scherm al in een tekstmode staat, dan stopt de routine zonder meer. Is dat niet het geval, dan wordt de inhoud van OLDSCR doorgegeven aan de standaardroutine CHGMOD.

```
adres     0848H
--------------
naam      CLS
in        vlag Z
uit       n.v.t.
wijzigt   AF,BC,DE,EI
```

Deze standaardroutine wist het scherm schoon in elke mode. Deze routine roept 0777H aan. Hier wordt eigenlijk het "CLS"-statement verwerkt. Omdat vlag NZ aangeeft dat er na dit statement nog tekst staat (niet toegelaten), doet de routine niets indien ze aangeroepen wordt met de Zero-vlag op 0.

```
adres     084FH
--------------
naam      CHGMOD
in        A = gewenste scherm-mode (0, 1, 2 of 3)
uit       n.v.t.
wijzigt   AF,BC,DE,HL,EI
```

Met deze standaardroutine wordt een bepaalde scherm-mode ingesteld. De inhoud van register A wordt bekeken en vervolgens wordt de controle overgedragen aan de standaardroutine INITXT, INIT32, INIGRP of INIMLT.

```
adres     085DH
--------------
naam      LPTOUT
in        A = te printen karakter
uit       vlag C indien op CTRL-STOP gedrukt werd
wijzigt   AF
```

Deze standaard routine stuurt een karakter naar de printer via de Centronics-poort. De toestand van de printer wordt continu onderzocht, via de standaardroutine LPTSTT, tot hij klaar is voor ontvangst. Dan wordt het karakter naar de Centronics Datapoort geschreven (In/Out-poort 91H) en wordt het Strobe-signaal van de Centronics Status-poort (In/Out-poort 90H) kortstondig omlaag gehaald. De BREAKX-routine wordt gebruikt om te testen of "CTRL/STOP" werd ingedrukt, terwijl de printer bezet is. Werd die toets-combinatie ingedrukt, dan wordt een "CR"-code (ODH) naar de Centronics Datapoort geschreven, om de regelbuffer van de printer leeg te maken, en de routine stopt met vlag C.

adres 0884H

---

naam LPTSTT
in n.v.t.
uit A - 0 en vlag Z als de printer bezet is
wijzigt AF

Deze standaardroutine onderzoekt het BUSY-signaal van de Centronics status-poort. In/Out-poort 90H wordt uitgelezen en bit 1 wordt bekeken. Is dit 0, dan is de printer klaar, is het 1 dan is hij bezet.

adres 088EH

---

naam POSIT
in H - kolom, L - rij
uit n.v.t.
wijzigt AF, EI

Deze standaardroutine bepaalt de coördinaten van de cursor. Die coördinaten worden aan de standaardroutine OUTDO doorgegeven in deze volgorde : "ESC", "Y", rij+1FH, kolom+1FH. Merk op dat de "home"-positie van het BIOS 1,1 is in plaats van 0,0 zoals de BASIC Interpreter gebruikt.

adres 089DH

---

naam CNVCHR
in A - karakter
uit vlag Z,NC - header; vlag NZ,C - grafisch; Z,C - normaal
wijzigt AF

Deze standaardroutine controleert karakters met grafische headers en zet ze indien nodig om. Normaal worden karakters met een code beneden 20H door de output-verwerkers geïnterpreteerd als controlecodes. Een karaktercode in dit gebied kan als een schrijfbaar karakter behandeld worden door het te doen voorafgaan van een controlecode 01H (grafische header) en 40H bij zijn code op te tellen. Om te vermijden dat karaktercode ODH als een "CR"-code wordt geïnterpreteerd, moet het naar de output-verwerkers gestuurd worden als 01H, 4DH. Deze routine wordt gebruikt door de outputverwerkers, zoals de standaardroutine CHPUT, om naar zulke groepen karaktercodes te zoeken.

Is het karakter een grafische header, dan wordt GRPHED op 1 gezet en stopt de routine, zoniet wordt GRPHED 0. Ligt de code

van het karakter buiten het bereik 40H tot 5FH, dan gebeurt er niets mee. Ligt het binnen dat bereik, en GRPHED bevat 1 (waarmee gesignaleerd wordt dat ervoor een grafische header kwam) dan wordt het omgezet door er 40H van af te trekken.

adres 08BCH

---

naam CHPUT
in A = karakter
uit n.v.t.
wijzigt EI

Deze standaardroutine zet een karakter op het scherm in 40x24 of 32x24 tekstmode. Eerst wordt SCRMOD bekeken en indien de Video Display Processor in grafische- of veelkleurenmode staat, stopt de routine zonder iets te doen. Is dit niet zo, dan wordt de cursor weggehaald (0A2EH), het karakter gedecodeerd (08DFH) en de cursor teruggezet (09E1H). Tenslotte wordt de kolompositie van de cursor in TTYPOS gezet, voor gebruik door het "PRINT"-statement, en de routine stopt.

adres 08DFH

Deze routine wordt door de standaardroutine CHPUT gebruikt om een karakter te decoderen, en dan het nodige te doen. Eerst wordt de standaardroutine CNVCHR aangeroepen om te kijken of het een grafisch karakter is. Is het karakter een grafische header (01H), dan stopt de routine zonder meer. Is het karakter een omgezet grafisch karakter, dan wordt het gedeelte waar de controlecodes gedecodeerd worden, overgeslagen. Zoniet, wordt in ESCCNT gekeken of voordien een "ESC" (1BH)-code werd ontvangen. Is dit zo, dan wordt de controle overgedragen aan de "ESC"-verwerker (098FH). Was dat niet het geval, dan wordt gecontroleerd of het karakter een code heeft onder 20H. Zo ja, wordt verder gegaan met de controlecode-verwerker (0914H). Is dat niet zo, dan wordt bekeken of het om een "DEL" (7FH)-code gaat. Zo ja, wordt verder gegaan in de wis-routine (0AE3H).

In de veronderstelling dat het karakter schrijfbaar is, worden de coördinaten van de cursor opgehaald uit CSRY en CSRX, en in registerpaar HL gezet (H = kolom, L = rij). Die coördinaten worden vervolgens omgezet naar een adres in de Namentabel van de Video Display Processor en het karakter wordt daar gezet (0BE6H). De positie van de cursor-kolom wordt opgehoogd (0A44H) en, indien daardoor de uiterst rechtse kolom niet overschreden werd, stopt de routine. Was dat wel het geval, dan wordt de overeenkomstige plaats in LINTTB op nul gezet, om aan te duiden dat die logische regel langer is dan een schermregel; het kolom-nummer wordt 1 gemaakt en er wordt een "LF" uitgevoerd (een regel lager op het scherm).

adres 0908H

Deze routine voert de LineFeed uit voor de controlecode-verwerker van de standaardroutine CHPUT. De cursor-rij wordt opgehoogd (0A61H) en indien die de onderste rij niet overschrijdt, stopt de routine. Zoniet, wordt het scherm naar boven gescrolld en de laagste regel gewist (0A88H).

adres 0914H

Hier worden de controlecodes verwerkt voor de standaardroutine CHPUT. De tabel vanaf adres 092FH wordt doorzocht tot de karaktercode wordt gevonden, en de controle wordt overgedragen aan de routine op het bijbehorende adres.

adres 092FH TABEL

Hier begint een tabel met controlecodes, met het bijbehorend adres, dat door de standaardroutine CHPUT herkend wordt :

CODE	NAAR	FUNCTIE
07H	1113H	BELL, doe "beep"
08H	0A4CH	BS, cursor naar links
09H	0A71H	TAB, cursor naar volgende TAB-positie
0AH	0908H	LF, cursor een regel lager
0BH	0A7FH	HOME, cursor naar 0,0
0CH	077EH	FORMFEED, wis scherm en cursor naar 0,0
0DH	0A81H	CR, cursor naar uiterst linkse kolom
1BH	0989H	ESC, begin van een "ESCAPE-groep"
1CH	0A5BH	RIGHT, cursor naar rechts
1DH	0A4CH	LEFT, cursor naar links
1EH	0A57H	UP, cursor naar omhoog
1FH	0A61H	DOWN, cursor naar omlaag

adres 0953H TABEL

Op dit adres begint een tabel met "ESC"-controlecodes, met voor elke code een bijbehorend adres dat door de standaardroutine CHPUT herkend wordt.

CODE	NAAR	FUNCTIE
6AH	077EH	ESC,"j", wis scherm en cursor op 0,0
45H	077EH	ESC,"E", wis scherm en cursor op 0,0
4BH	0AEEH	ESC,"K", wis tot eind van de regel
4AH	0B05H	ESC,"J", wis tot eind van het scherm
6CH	0AECH	ESC,"l", wis regel
4CH	0AB4H	ESC,"L", voeg regel tussen
4DH	0A85H	ESC,"M", wis regel
59H	0986H	ESC,"Y", vul coördinaten van cursor in
41H	0A57H	ESC,"A", cursor omhoog
42H	0A61H	ESC,"B", cursor omlaag
43H	0A44H	ESC,"C", cursor naar rechts
44H	0A55H	ESC,"D", cursor naar links
48H	0A7FH	ESC,"H", cursor naar 0,0
78H	0980H	ESC,"x", verander cursor
79H	0983H	ESC,"y", verander cursor

adres 0980H

Deze routine voert de "ESC "x""-bewerking uit voor de controlecode-verwerker van de standaardroutine CHPUT. In ESCCNT wordt 01H gezet, om aan te geven dat het volgende karakter een parameter is.

adres 0983H

Deze routine voert de "ESC "y""-bewerking uit voor de controlecode-verwerker van de standaardroutine CHPUT. In ESCCNT wordt 02H gezet, om aan te geven dat het volgende karakter een parameter is.

adres 0986H

Deze routine voert de "ESC "Y""-bewerking uit voor de controlecode-verwerker van de standaardroutine CHPUT. In ESCCNT wordt 04H gezet, om aan te geven dat het volgende karakter een parameter is.

adres 0989H

Deze routine voert de "ESC"-bewerking uit voor de controlecode-verwerker van de standaardroutine CHPUT. In ESCCNT wordt FFH gezet, om aan te geven dat het volgende karakter het tweede controle - karakter is.

adres 098FH

Deze routine verwerkt de "ESC"-groepen voor de standaardroutine CHPUT. Bevat ESCCNT FFH, dan is het behandelde karakter het tweede controlekarakter, en wordt de controle overgedragen aan de controlecode-verwerker (0919H), die de "ESC"-codetabel op 0953H doorzoekt.

Bevat ESCCNT 01H dat is het huidige karakter de enige parameter van de "ESC","x"-groep. Is de parameter "4" (34H) dan wordt 00H in CSTYLE gezet, wat een blokvormige cursor geeft. Als de parameter "5" (35H) is, wordt 00H in CSRSW gezet, wat de cursor onzichtbaar maakt.

Indien ESCCNT 02H bevat, is het huidige karakter de enige parameter in de "ESC","y"-groep. Als die parameter "4" (34H) is, wordt in CSTYLE 01H gezet, waardoor de cursor een half blokje wordt. Is de parameter "5" (35H) dat wordt in CSRSW 01H gezet, waardoor de cursor normaal zichtbaar wordt.

Bevat ESCCNT 04H dat is het huidige karakter de eerste parameter van de "ESC","Y"-groep, en is dit het lijn-coördinaat. Er wordt 1FH van afgetrokken en het resultaat wordt in CSRY gezet. ESCCNT wordt 03H.

Bevat ESCCNT 03H, dan is het huidige karakter de tweede parameter van de "ESC","Y"-groep, en is dit het kolom-coördinaat. Er wordt 1FH van afgetrokken en het resultaat wordt in CSRX gezet.

adres 09DAH

Deze routine wordt, bijvoorbeeld door de standaardroutine CHGET, gebruikt om de cursor op het scherm te zetten, terwijl hij normaal niet zichtbaar is. Als CSRSW verschilt van nul, stopt de routine zonder meer. Zoniet, wordt de cursor op het scherm gezet (09E6H).

adres 09E1H

Deze routine wordt, bijvoorbeeld door de standaardroutine CHPUT, gebruikt om de cursor op het scherm te zetten, terwijl hij normaal zichtbaar is. Als CSRSW nul is, stopt de routine zonder meer. SCRMOD wordt bekeken en indien de Video Display Processor in grafische of veelkleurenmode staat, stopt de routine zonder meer. Zoniet, worden de cursor-coördinaten omgerekend naar een adres in de Namentabel van de Video Display Processor, het karakter op die plaats wordt uitgelezen (0BD8H) en in CURSAV gezet.

Het 8-bytes pixelpatroon van het karakter wordt uit de Karaktertabel van de Video Display Processor gelezen, in de LINWRK-buffer (0BA5H). Het pixelpatroon wordt geïnverteerd, van de acht bytes indien CSTYLE een blokvormige cursor aangeeft, of enkel de onderste vier indien CSTYLE een halve cursor specificeert. Dan wordt het pixelpatroon teruggezet op de plaats van karaktercode 255 in de Karaktertabel van de Video Display Processor (0BBEH). Dan wordt karaktercode 255 op de plaats in de Namentabel gezet, waar momenteel de cursor staat (0BE6H) en de routine stopt.

Deze manier om de cursor op het scherm te zetten, door gebruik te maken van karaktercode 255, kan, onder bepaalde omstandigheden, eigenaardige resultaten opleveren. Een demonstratie daarvan kunnen we zien door de BASIC regel : "FOR N=1 TO 100: PRINT CHR$(255);: NEXT" in te voeren, en dan op de "cursor omhoog"-toets te drukken.

adres 0A27H

Deze routine wordt, bijvoorbeeld door de standaardroutine CHGET, gebruikt om de cursor van het scherm te halen terwijl hij normaal onzichtbaar is. Bevat CSRSR niet nul, dan stopt de routine zonder meer. Zoniet, wordt de cursor van het scherm gehaald (0A33H).

adres 0A2EH

Deze routine wordt, bijvoorbeeld door de standaardroutine CHPUT, gebruikt om de cursor van het scherm te halen terwijl hij normaal zichtbaar is. Bevat CSRSW nul, dan stopt de routine zonder meer. SCRMOD wordt gecontroleerd en, wanneer het scherm in grafische- of veelkleurenmode staat, stopt de routine zonder meer. Zoniet, worden de cursor-coördinaten omgerekend naar een adres in de Namentabel van de Video Display Processor en het karakter dat in CURSAV staat wordt op die plaats gezet (0BE6H).

adres 0A44H

Deze routine voert de "ESC","C"-bewerking uit voor de controlecode-verwerker van de standaardroutine CHPUT. Indien het kolom-coördinaat van de cursor de meest rechtse kolom aangeeft (dat staat in LINLEN), dan stopt de routine zonder meer. Zoniet, wordt het kolom-coördinaat opgehoogd, en CSRX aangepast.

adres 0A4CH

Deze routine voert de "BS"/LINKS-bewerking uit voor de controlecode-verwerker van de standaardroutine CHPUT. Het kolom-coördinaat van de cursor wordt verlaagd en CSRX aangepast. Wanneer het kolom-coördinaat lager werd dan de uiterst linkse positie, wordt hij op de uiterst rechtse positie gezet (uit LINLEN) en wordt een "OMHOOG"-bewerking uitgevoerd.

adres 0A55H

Deze routine voert de "ESC","D"-bewerking uit voor de controlecode-verwerker van de standaardroutine CHPUT. Staat de cursor al op de meest linkse positie, dan stopt de routine zonder meer. Zoniet, wordt het kolom-coördinaat verlaagd en CSRX aangepast.

adres 0A57H

Deze routine voert de "ESC","A"/"OMHOOG"-bewerking uit voor de controlecode-verwerker van de standaardroutine CHPUT. Indien de cursor al op de bovenste rij staat, stopt de routine zonder meer. Zoniet, wordt het rij-coördinaat verlaagd en CSRY aangepast.

adres 0A5BH

Deze routine voert de "RECHTS"-bewerking uit voor de controlecode-verwerker van de standaardroutine CHPUT. Het kolom-coördinaat van de cursor wordt opgehoogd en CSRX aangepast. Gaat het coördinaat verder dan de uiterst rechtse positie, die in LINLEN staat, dan wordt hij op de meest linkse positie gezet (01H) en wordt er een "OMLAAG"-bewerking uitgevoerd.

adres 0A61H

Deze routine voert de "ESC","B"/OMLAAG"-bewerking uit voor de controlecode-verwerker van de standaardroutine CHPUT. Staat de cursor al op de onderste rij (aangegeven door CTRCNT en CNSDFG (0C32H)), dan stopt de routine zonder meer. Zoniet, wordt het rij-coördinaat opgehoogd en CSRY aangepast.

adres 0A71H

Deze routine voert de "TAB"-bewerking uit voor de controlecode-verwerker van de standaardroutine CHPUT. Er worden ASCII-spaties naar het scherm gestuurd (08DFH) tot CSRX een veelvoud van 8 plus 1 is (BIOS-kolommen 1,9,17,25,33).

adres 0A7FH

Deze routine voert de "ESC","H"/"HOME"-bewerking uit voor de controlecode-verwerker van de standaardroutine CHPUT. CSRX en CSRY worden gewoon op 1 en 1 gezet. Het coördinatensysteem van de cursor, zoals gehanteerd door het ROM BIOS is functioneel gelijk aan het systeem dat de BASIC Interpreter gebruikt, maar nummert de rijen op het scherm van 1 tot 24 en de kolommen van 1 tot 32 of 40.

adres 0A81H

Deze routine voert de "CR"-bewerking uit voor de controlecode-verwerker van de standaardroutine CHPUT. CSRX wordt gewoon op 1 gezet.

adres 0A85H

Deze routine voert de "ESC","M"-bewerking uit voor de controlecode-verwerker van de standaardroutine CHPUT. Eerst wordt een "CR" uitgevoerd om het kolom-coördinaat van de cursor op de uiterst linkse positie te zeten. Dan wordt het aantal regels vanaf de huidige regel tot de onderkant van het scherm bepaald. Is dat aantal nul, dan wordt gewoon de huidige regel gewist (0ACEH). Het aantal regels wordt eerst gebruikt om het betreffende stuk van LINTTB 1 byte verder te scrollen, en dan om het overeenkomstige stuk van het scherm een regel tegelijk omhoog te scrollen. Te beginnen met de regel onder de huidige regel, wordt elke regel gecopieerd vanuit de Namentabel van de Video Display Processor naar de LINWRK-buffer (0BAAH), en dan weer naar de Namentabel, maar 1 regel hoger (0BC3H). Tenslotte wordt de laagste regel op het scherm gewist. (0AECH).

adres 0AB4H

Deze routine voert de "ESC","L"-bewerking uit voor de controlecode-verwerker van de standaardroutine CHPUT. Eerst wordt een "CR" uitgevoerd, om het kolom-coördinaat van de cursor op de uiterst linkse positie te zetten. Dan wordt het aantal regels vanaf de huidige stand tot de onderkant van het scherm bepaald. Als dat aantal nul is, wordt gewoon de huidige regel gewist (0AECH). Eerst wordt het aantal regels gebruikt om het overeenkomstige stuk van LINTTB naar beneden te scrollen met 1 byte. Dan wordt het gebruikt om het betreffende schermdeel een regel tegelijk naar beneden te scrollen. Te beginnen bij de voorlaatste regel op het scherm, wordt elke regel gecopieerd vanuit de Namentabel van de Video Display Processor naar de LINWRK-buffer (0BAAH), en dan weer naar de Namentabel, maar een regel lager (0B3CH). Tenslotte wordt de huidige regel gewist (0AECH).

adres 0AE3H

Deze routine voert de "DELete"-bewerking uit voor de controlecode-verwerker van de standaardroutine CHPUT. Eerst wordt een "LINKS"-bewerking uitgevoerd. Kan dit niet, omdat de cursor al op de uiterste positie staat, dan stopt de routine zonder meer. In het andere geval wordt een spatie geschreven in de Namentabel van de Video Display Processor, op de plaats van de cursor (0BE6H).

adres 0AECH

Deze routine voert de "ESC","l"-bewerking uit voor de controlecode-verwerker van de standaardroutine CHPUT. Het kolom-coördinaat van de cursor wordt op 01H gezet, en de routine gaat verder met de "ESC", "K"-routine.

adres OAEEH

Deze routine voert de "ESC","K"-bewerking uit voor de controlecode-verwerker van de standaardroutine CHPUT. De overeenkomstige plaats in LINTTB wordt eerst niet-nul gemaakt, om aan te geven dat de logische regel niet uitgebreid werd (OC29H). De cursor-coördinaten worden omgezet in een adres in de Namentabel van de Video Display Processor (OBF2H) en de Video Display Processor klaargemaakt vor een schrijf-operatie via SETWRT. Dan worden spaties rechtstreeks naar de Datapoort van de Video Display Processor geschreven tot de uiterst rechtse kolom (aangegeven door LINLEN), bereikt wordt.

adres OB05H

Deze routine voert de "ESC","J"-bewerking uit voor de controlecode-verwerker van de standaardroutine CHPUT. Te beginnen met de huidige regel tot het einde van het scherm, worden opeenvolgende "ESC","K"-bewerkingen uitgevoerd.

adres OB15H
----

naam ERAFNK
in n.v.t.
uit n.v.t.
wijzigt AF,DE,EI

Deze standaardroutine zorgt ervoor dat de tekst van de functie-toetsen niet meer op het scherm komt. Eerst wordt nul in CNSDFG gezet en indien de Video Display Processor in grafische of veelkleuren-mode staat, stopt de routine zonder meer. Staat de Video Display Processor in 40x24 tekst mode of 32x24 tekst mode, dan wordt de laatste schermregel gewist (OAECH).

adres OB26H
----

naam FNKSB
in n.v.t.
uit n.v.t.
wijzigt AF,BC,DE,EI

Deze standaardroutine zorgt ervoor dat de tekst van de functietoetsen op het scherm staat, als dat toegelaten is. Als CNSDFG nul bevat, stopt de routine zonder meer. Zoniet, gaat de routine verder met de standaardroutine DSPFNK.

adres OB2BH
----

naam DSPFNK
in n.v.t.
uit n.v.t.
wijzigt AF,BC,DE,EI

Deze standaardroutine maakt de teksten van de functietoetsen op het scherm zichtbaar. CNSDFG wordt op 255 gezet. Indien de Video Display Processor in grafische- of veelkleurenmode staat, stopt de routine zonder meer. Zoniet, wordt het rij-coördinaat van de cursor bekeken en indien de cursor op de laatste regel staat, wordt een LineFeed code (OAH) naar de standaardroutine OUTDO

gestuurd, om het scherm omhoog te scrollen.
Dan wordt registerpaar HL ingevuld met het startadres van de teksten die bij de functietoetsen horen, in gewone stand ofwel in de SHIFT-stand, wanneer de SHIFT-toets wordt ingedrukt. Van LINLEN wordt vier afgetrokken, om mimimum 1 spatie tussen de diverse teksten te laten, en gedeeld door vijf om de ruimte te bepalen die elke string zal innemen. Vervolgens worden de opeenvolgende karakters uit de string gelezen, gecontroleerd op grafische headers via de standaardroutine CNVCHR en in de LINWRK-buffer geplaatst tot er ofwel geen karakters meer in de string voorkomen, ofwel de beschikbare ruimte in de zone gevuld is. Zijn de vijf strings volledig, dan wordt de inhoud van de LINWRK-buffer naar de laatste rij in de Namentabel van de Video Display Processor geschreven (OBC3H).

adres OB9CH

Deze routine wordt gebruikt door de andere routines die verband houden met het op het scherm zetten van de tekst van de functietoetsen. De inhoud van register A wordt in CNSDFG gezet, SCRMOD gecontroleerd, en de routine keert terug met vlag NC indien het scherm in grafische of veelkleurenmode staat.

adres OBA5H

Deze routine copieert acht bytes van VRAM naar de LINWRK-buffer. Het adres in VRAM staat in registerpaar HL.

adres OBAAH

Deze routine copieert een volledige regel karakters, waarvan de lengte bepaald wordt door LINLEN, vanuit VRAM naar de LINWRK-buffer. Het lijn-coördinaat van de cursor staat in register L.

adres OBBEH

Deze routine copieert acht bytes van de LINWRK-buffer naar VRAM. Het adres in VRAM staat in registerpaar HL.

adres OBC3H

Deze routine copieert een volledige regel karakters, waarvan de lengte door LINLEN bepaald wordt, van de LINWRK-buffer naar VRAM. Het lijn-coördinaat van de cursor staat in register L.

adres OBD8H

Deze routine leest één byte uit VRAM in register C. Het kolomcoördinaat staat in register H, het lijn-coördinaat in register L.

adres OBE6H

Deze routine bepaalt welk adres in de Namentabel van de Video Display Processor hoort bij een stel scherm-coördinaten. Het kolom-coördinaat staat in register H, en het lijn-coördinaat in register L. Het adres staat, op het einde van de routine, in registerpaar HL.

Het lijn-coördinaat wordt eerst met 32 of 40 vermenigvuldigd.

afhankelijk van de scherm-mode, en bij de kolom-coördinaat opgeteld. Dit resultaat wordt dan opgeteld bij het basisadres van de Namentabel van de Video Display Processor (uit NAMBAS), om een eerste adres te verkrijgen.

Omwille van de veranderlijke schermbreedte (in LINLEN), moet dit adres aangepast worden zodat het actieve gebied nog min of meer in het midden van het scherm staat. Het verschil tussen het "echte" aantal karakters per regel, 32 of 40, en de op dat moment gekozen schermbreedte, wordt gehalveerd en dan opgerond. Zo verkijgen we de aanpassing voor de linkse kolom. Bij een Engelse machine, waar in 40x24 tekst mode, 37 karakters gebruikt worden, blijven zo twee karakter-posities links en één rechts ongebruikt. Het statement : PRINT (41-WID)\2, waarbij WID de schermbreedte is, levert de aanpassing op voor de linker kolom in 40x24 tekst mode.

Hieronder volgt een BASIC programma dat hetzelfde doet als deze routine.

```
10 CPR=40:NAM=BASE(0):WID=PEEK(&^F3AE)
20 SCRMD=PEEK(&HFCAF):IF SCRMOD=0 THEN 40
30 CRP=32:NAM=BASE(5):WID=PEEK(&HF3AF)
40 LH=(CPR+1-WID)\2
50 ADDR=NAM+(ROW-1)*CPR+(COL-1)+LH
```

Dit programma werd geschreven om te werken met de RIJ en KOLOM-coordinaten van het ROM BIOS, waar de "HOME"-positie 1,1 is. Regel 50 kan vereenvoudigd worden, door de "-1"-factoren weg te halen, indien het coördinatenstelsel van de BASIC Interpreter gebruikt moet worden.

adres 0C1DH

Deze routine berekent het adres van de beschrijving in LINTTB van een bepaalde rij. Het rij-coördinaat staat in register L, het adres staat op het einde van de routine in registerpaar DE.

adres 0C29H

Deze routine zet het byte dat bij een bepaalde regel behoort in LINTTB, op een waarde verschillend van 0 wanneer ze op 0C29H aangeroepen wordt, en op 0 wanneer ze op 0C2AH wordt aangeroepen. Het rij-coördinaat staat in register L.

adres 0C32H

Met deze routine wordt berekend hoeveel regels er op een scherm kunnen. Het aantal wordt in register A gezet. Normaal zal dit 24 zijn, indien de tekst van de functietoetsen niet op het scherm staat, en 23 indien die tekst er wel staat. De grootte van het scherm wordt bepaald door CRTCNT, en kan vanuit BASIC gewijzigd worden, bijvoorbeeld met : POKE &HF3B1H,14:SCREEN 0.

adres 0C3CH
----

naam KEYINT
in n.v.t.
uit n.v.t.
wijzigt EI

Deze standaardroutine verwerkt de interrupts van de Z80. Die interrupts worden door de Video Display Processor elke 20 msec opgewekt (bij een Europese machine). Eerst wordt het statusregister van de Video Display Processor bekeken, en bit 7 gecontroleerd om zeker te zijn dat het om een interrupt gaat vanuit de Video Display Processor. Is dat niet zo, dan stopt de routine zonder meer. Zoniet, wordt de inhoud van het statusregister in STATFL gezet, en bit 5 gecontroleerd om te zien of er sprites overlappen. Is de coïncidentie-vlag geset, dan wordt het overeenkomstige byte in TRPTBL aangepast (OEF1H).

Vervolgens wordt INTCNT (de "INTERVAL"-teller) verlaagd. Wordt die teller nul, dan wordt de bijbehorende plaats in TRPTBL aangepast (OEF1H) en wordt de teller ingevuld met de inhoud van INTVAL.

JIFFY, de "TIME"-teller, wordt opgehoogd. Wanneer die zijn maximum bereikt, wordt die gewoon weer 0.

MUSICF wordt onderzocht, om te zien of een van de drie muziek-rijen die door het PLAY-statement worden aangemaakt, actief is. Voor elke actieve rij wordt de afhandelroutine aangeroepen (113BH) om het volgende pakket naar de PSG te schrijven.

Vervolgens wordt SCNCNT verlaagd met 1, om te zien of de joystick en het toetsenbord moeten worden afgetast. Is dat niet nodig, dan stopt de routine zonder meer. Die teller zorgt ervoor dat de verwerkingssnelheid verhoogd wordt, en dat kontaktdender-problemen zo weinig mogelijk voorkomen, door maar om de drie interupts een aftasting uit te voeren. In de veronderstelling dat er een aftasting moet gebeuren, wordt joystick-aansluiting 1 actief gemaakt en de twee vuurknop-bits uitgelezen (12C0H), gevolgd door dezelfde twee bits van joystick 2 (12C0H) en de spatietoets op rij 8 van het toetsenbord (1226H). Deze vijf inputs, die allemaal te maken hebben met het "STRIG"-statement, worden tot één byte verwerkt, waarin een bit 0 betekent : ingedrukt, en een bit 1 : niet ingedrukt.

7	6	5	4	3	2	1	0
joy 2 trg.B	joy 2 trg.A	joy 1 trg.B	joy 1 trg.A	0	0	0	space

Fig. 35 : "STRIG"-inputs

Deze uitlezing wordt vergeleken met de vorige uitlezing, die in TRGFLG staat, om een actief overgangsbyte te maken, en TRGFLG wordt ingevuld met de laatste uitlezing. Normaal is dit overgangsbyte 0, maar het bevat een "1"-bit op elke plaats waar een overgang van niet-ingedrukt naar ingedrukt heeft plaatsgehad. Dit byte wordt bit na bit onderzocht, en de overeenkomstige plaats in TRPTBL wordt aangepast voor elk actief apparaat (OEF1H).

Vervolgens wordt de komplete matrix van het toetsenbord

onderzocht, naar ingedrukte toetsen. Elke druk op een toets wordt vertaald in een code, en in KEYBUF gezet (OD12H). Blijkt, aan het einde van deze bewerking, dat KEYBUF leeg is, dan wordt REPCNT verlaagd met 1 om te zien of de auto-repeat periode voorbij is. Zoniet, stopt de routine. Was de periode wel voorbij, dan wordt in REPCNT de waarde gezet voor een snelle herhaling van toetsen (60 ms), de OLDKEY toetsenbord-tabel wordt geïnitialiseerd en het toetsenbord wordt nog eens afgetast (OD4EH). Wanneer tijdens deze aftasting een toets continu ingedrukt wordt, geeft hij telkens weer overgangen (zie hoger). Merk op, dat een toets alleen zal herhalen indien een toepassingsprogramma KEYBUF leeg houdt, door karakters uit de buffer te lezen. Nu eindigt de interrupt-routine.

adres   OD12H

Deze routine verzorgt een volledige aftasting van de toetsen-matrix voor de interupt-routine. Elk van de elf rijen wordt ingelezen via de PPI, en in opklimmende volgorde in NEWKEY gezet. ENSTOP wordt dan gecontroleerd om te zien of warme starts mogelijk zijn. Wanneer de inhoud daarvan niet nul is, en de toetsen : "CODE", "GRPH", "CTRL" en "SHIFT" tegelijk worden ingedrukt, wordt de controle aan de BASIC Interpreter overgedragen (409BH) via de standaardroutine CALBAS. Deze voorziening is nuttig, omdat ze toelaat om zelfs een machinetaal-programma te beëindigen, zolang de interrupts worden verwerkt.

De inhoud van NEWKEY wordt vergeleken met de inhoud van OLDKEY, die het resultaat van de vorige aftasting bevat. Wordt een verandering opgemerkt, dan wordt REPCNT geladen met de beginwaarde voor de wachttijd tussen het afdrukken van twee toetsen bij auto-repeat (780 msec). De uitlezing van elke rij in NEWKEY wordt dan vergeleken met de overeenkomstige in OLDKEY, waarmee een actief overgangs-byte wordt aangemaakt, en OLDKEY wordt aan de laatste uitlezing aangepast. Dit overgangsbyte is normaal 0, maar bevat een "1"-bit op elke positie waar een overgang van niet-ingedrukte toets naar ingedrukte toets plaatsvond. Bevat de rij zo een overgang, dan wordt die gedecodeerd en als een toets-code in KEYBUF gezet (OD89H). Zijn de elf rijen afgewerkt, dan controleert de routine nog of er in KEYBUF karakters staan (door GETPNT van PUTPNT af te trekken), en stopt dan.

adres   OD6AH

```
naam     CHSNS
in       n.v.t.
uit      vlag NZ indien in KEYBUF karakters staan
wijzigt  AF,EI
```

Deze standaardroutine controleert of KEYBUF karakters bevat. Staat het scherm in grafische- of veelkleurenmode, dan wordt GETPNT van PUTPNT afgetrokken (OD62H) en de routine stopt hier. Staat het scherm in de 40x24 of 32x24 tekst mode, dan wordt eveneens de toestand van de SHIFT-toets bekeken en indien die sinds de laatste keer veranderd is, wordt via de standaardroutine DSPFNK de tekst van de andere vijf functietoetsen op het scherm afgedrukt.

adres   0D89H

Deze routine zet elk "1"-bit in een overgangsbyte om in een toets-code. Eerst wordt het bit omgezet in een toets-nummer, bepaald door zijn positie in de matrix van het toetsenbord :

7 (07H)	6 (06H)	5 (05H)	4 (04H)	3 (03H)	2 (02H)	1 (01H)	0 (00H)	RIJ 0
; (0FH)	] (0EH)	[(0DH)	\ (0CH)	= (0BH)	- (0AH)	9 (09H)	8 (08H)	RIJ 1
B (17H)	A (16H)	£ (15H)	/ (14H)	. (13H)	, (12H)	' (11H)	' (10H)	RIJ 2
J (1FH)	I (1EH)	H (1DH)	G (1CH)	F (1BH)	E (1AH)	D (19H)	C (18H)	RIJ 3
R (27H)	Q (26H)	P (25H)	O (24H)	N (23H)	M (22H)	L (21H)	K (20H)	RIJ 4
Z (2FH)	Y (2EH)	X (2DH)	W (2CH)	V (2BH)	U (2AH)	T (29H)	S (28H)	RIJ 5
F3 (37H)	F2 (36H)	F1 (35H)	CODE (34H)	CAP (33H)	GRAPH (32H)	CTRL (31H)	SHIFT (30H)	RIJ 6
CR (3FH)	SEL (3EH)	BS (3DH)	STOP (3CH)	TAB (3BH)	ESC (3AH)	F5 (39H)	F4 (38H)	RIJ 7
RECHTS (47H)	OMLAAG (46H)	OMHOOG (45H)	LINKS (44H)	DEL (43H)	INS (42H)	HOME (41H)	SPACE (40H)	RIJ 8
4 (4FH)	3 (4EH)	2 (4DH)	1 (4CH)	0 (4BH)	(4AH)	(49H)	(48H)	RIJ 9
. (57H)	, (56H)	- (55H)	9 (54H)	8 (53H)	7 (52H)	6 (51H)	5 (50H)	RIJ10
7	6	5	4	3	2	1	0	KOLOM

Fig. 36 : nummers van de toetsen

Het nummer van de toets wordt vervolgens in een toets-code omgezet die in KEYBUF geplaatst wordt (1021H). Wanneer alle acht bits verwerkt zijn, stopt de routine.

adres   0DA5H   TABEL

Op dit adres start een tabel met de toets-nummers 00H tot 2FH, voor verschillende combinaties van de controletoetsen. Een nul in die tabel betekent dat indrukken van die toetsencombinatie, geen toets-code oplevert.

NORMAAL	37H	36H	35H	34H	33H	32H	31H	30H	RIJ 0
	3BH	5DH	5BH	5CH	3DH	2DH	39H	38H	RIJ 1
	62H	61H	9CH	2FH	2EH	2CH	60H	27H	RIJ 2
	6AH	69H	68H	67H	66H	65H	64H	63H	RIJ 3
	72H	71H	70H	6FH	6EH	6DH	6CH	6BH	RIJ 4
	7AH	79H	78H	77H	76H	75H	74H	73H	RIJ 5
SHIFT	26H	5EH	25H	24H	23H	40H	21H	29H	RIJ 0
	3AH	7DH	7BH	7CH	2BH	5FH	28H	2AH	RIJ 1
	42H	41H	9CH	3FH	3EH	3CH	7EH	22H	RIJ 2
	4AH	49H	48H	47H	46H	45H	44H	43H	RIJ 3
	52H	51H	50H	4FH	4EH	4DH	4CH	4BH	RIJ 4
	5AH	59H	58H	57H	56H	55H	54H	53H	RIJ 5
GRAPH	FBH	F4H	BDH	EFH	BAH	ABH	ACH	09H	RIJ 0
	06H	0DH	01H	1EH	F1H	17H	07H	ECH	RIJ 1
	11H	C4H	9CH	1DH	F2H	F3H	BBH	95H	RIJ 2
	C6H	DCH	13H	15H	14H	CDH	C7H	BCH	RIJ 3
	18H	CCH	DBH	C2H	1BH	0BH	C8H	DDH	RIJ 4
	0FH	19H	1CH	CFH	1AH	C0H	12H	D2H	RIJ 5
SHIFT+ GRAPH	00H	F5H	00H	00H	FCH	FDH	00H	0AH	RIJ 0
	04H	0EH	02H	16H	F0H	1FH	08H	00H	RIJ 1
	00H	FEH	9CH	F6H	AFH	AEH	F7H	03H	RIJ 2
	CAH	DFH	D6H	10H	D4H	CEH	C1H	FAH	RIJ 3
	A9H	CBH	D7H	C3H	D3H	0CH	C9H	DEH	RIJ 4
	F8H	AAH	F9H	D0H	D5H	C5H	00H	D1H	RIJ 5
CODE	E1H	E0H	98H	9BH	BFH	D9H	9FH	EBH	RIJ 0
	B7H	DAH	EDH	9CH	E9H	EEH	87H	E7H	RIJ 1
	97H	84H	9CH	A7H	A6H	86H	E5H	B9H	RIJ 2
	91H	A1H	B1H	81H	94H	8CH	8BH	8DH	RIJ 3
	93H	83H	A3H	A2H	A4H	E6H	B5H	B3H	RIJ 4
	85H	A0H	8AH	88H	95H	82H	96H	89H	RIJ 5
SHIFT+ CODE	00H	00H	9DH	9CH	BEH	9EH	ADH	D8H	RIJ 0
	B6H	EAH	E8H	00H	00H	00H	80H	E2H	RIJ 1
	00H	8EH	9CH	A8H	00H	8FH	E4H	B8H	RIJ 2
	92H	00H	B0H	9AH	99H	00H	00H	00H	RIJ 3
	00H	00H	E3H	00H	A5H	00H	B4H	B2H	RIJ 4
	00H	00H	00H	00H	00H	90H	00H	00H	RIJ 5
	7	6	5	4	3	2	1	0	KOLOM

adres ØEC5H

De controle wordt aan deze routine overgegeven, op het einde van de decodering van de functietoetsen (ØFC3H). De overeenkomstige plaats in FNKFLG wordt eerst gecontroleerd, om te zien of die bepaalde functietoets verbonden is met een "ON KEY GOSUB"-statement. Is dat zo, en geeft CURLIN tegelijk aan dat de BASIC Interpreter een programma aan het verwerken is, dan wordt de overeenkomstige plaats in TRPTBL aangepast (ØEF1H) en de routine stopt hier.

Is de toets niet verbonden met een "ON KEY GOSUB"-statement, of is de Interpreter niet met een programma bezig, dan wordt de tekst-string die bij die toets hoort, opgehaald. Het toetsnummer wordt met zestien vermenigvuldigd (elke string is 16 tekens lang) en het resultaat wordt opgeteld bij het startadres van de reeks strings in het Werkgebied. De opeenvolgende tekens uit die string worden in KEYBUF geplaatst (ØF55H) tot het einde (een Ø-byte).

adres ØEF1H

Deze routine dient om TRPTBL aan te passen, wanneer een of ander apparaat een interrupt heeft gegenereerd tijdens een BASIC programma. Bij het begin bevat registerpaar HL het adres in de tabel van het statusbyte dat bij het apparaat hoort. Eerst wordt bit Ø van het statusbyte gecontroleerd, en als het apparaat niet "AAN" is, stopt de routine zonder meer. Bit 2, de "Event"-vlag, wordt vervolgens bekeken. Is deze bit 1, dan stopt de routine; zoniet, wordt het bit geset om aan te geven dat er een "Event" heeft plaatsgehad. Bit 1, de "STOP"-vlag wordt nu bekeken. Werd het apparaat onderbroken, dan gebeurt er niets meer. Is dat niet het geval, dan wordt ONGSBF opgehoogd, waardoor de Verwerkings-lus van de Interpreter weet dat het "Event" nu afgehandeld dient te worden.

adres ØFØ6H

Dit stukje van de toets-decoder verwerkt enkel de "HOME"-toets. De toestand van de "SHIFT"-toets wordt bepaald via rij 6 van NEWKEY en al naargelang het resultaat daarvan, wordt de toetscode voor "HOME" (ØBH) of "CLS" (ØCH) in KEYBUF gezet (ØF55H).

adres ØF1ØH

Dit stuk van de toets-decoder verwerkt de toetsen met nummers tussen 3ØH en 57H, behalve "CAP", "F1" tot "F5", "STOP" en "HOME". Het nummer van de toets wordt gebruikt om de toetscode op te zoeken in de tabel op 1Ø33H, en die code wordt in KEYBUF gezet (ØF55H).

adres ØF1FH

Dit stuk van de toets-decoder verwerkt de "dode" toets, die op Europese machines voorkomt. Op Engelse machines geeft de toets in rij twee, kolom vijf, altijd de toetscode voor het Pond-teken (9CH), zoals te zien is in de tabel op adres ØDA5H. Op Europese machines staat op dezelfde plaatsen de code 255. Dit is alleen

een vlag, om aan te geven dat de volgende toets, indien het een klinker is, gewijzigd dient te worden in een grafisch karakter met een accent.

De toestand van de "SHIFT" en "CODE"-toetsen wordt uitgelezen via rij 6 van NEWKEY en één van de volgende posities in KANAST : 1 - DOOD, 2 - DOOD+SHIFT, 3 - DOOD+CODE, 4 - DOOD+SHIFT+CODE.

adres ØF36H

Dit stukje van de toets-decoder verwerkt de "CAP"-toets. De huidige stand van CAPST wordt omgekeerd, en de controle wordt overgedragen aan de standaardroutine CHGCAP.

adres ØF3DH

naam	CHGCAP
in	A = AAN/UIT-schakelaar
uit	n.v.t.
wijzigt	AF

Deze standaardroutine zet de LED bij de "CAP"-toets aan of uit, al naargelang de inhoud van register A : Ø - aan, 1 - uit. Om de LED te schakelen wordt de Mode-poort van de PPI gebruikt, die een bit kan setten en resetten. Omdat CAPST niet wordt gewijzigd, heeft deze routine geen invloed op de karakters die door een druk op een toets worden geproduceerd : indien dat voordien hoofdletters waren, blijven dat hoofdletters, en kleine letters blijven kleine letters.

adres ØF46H

Dit stukje van de toets-decoder verwerkt de "STOP"-toets. De toestand van de "CTRL"-toets wordt bepaald via rij 6 van NEWKEY en de code voor "STOP" (Ø4H) of voor "CTRL/STOP" (Ø3H) wordt gegenereerd. Was het de code voor "CTRL/STOP", dan wordt die gecopieerd in INTFLG, waar de standaardroutine ISCNTC ze later zal ophalen, en vervolgens in KEYBUF (ØF55H). Was het de "STOP"-code, dan wordt ze ook in INTFLG gezet, maar niet in KEYBUF. Er wordt enkel een klik geproduceerd (ØF64H). Dit betekent dat een toepassingsprogramma de "STOP"-toets niet via de ROM BIOS-routines kan aftasten.

adres ØF55H

Dit stuk van de toets-decoder zet een toetscode in KEYBUF en produceert een hoorbare klik. Eerst wordt het juiste adres in de buffer uit PUTPNT gelezen, en op dit adres wordt de code gezet. Vervolgens wordt het adres opgehoogd (105BH). Indien dat adres doortelt, en hetzelfde wordt als GETPNT, dan stopt de routine hier : de buffer is vol. Is dat niet zo, dan wordt in PUTPNT het nieuwe adres ingevuld.

CLIKSW en CLIKFL worden beide bekeken, om te zien of er een klik geproduceerd moet worden. CLIKSW is een algemene mag/mag niet-schakelaar, terwijl CLIKFL voorkomt dat er meerdere klikjes hoorbaar zijn bij het indrukken van een functietoets. In het geval dat er een klik moet worden geproduceerd, wordt via de Mode-poort van de PPI de Key Click Output geset, en na 5Ø µsec wordt de controle overgegeven aan de standaardroutine CHGSND.

adres   ØF7AH

---

naam   CHGSND
in   A = aan/uit-schakelaar
uit   n.v.t.
wijzigt   AF

Deze standaardroutine set of reset de Key Click Output via de Mode poort van de PPI : Ø = reset, 1 = set. Deze audio-output is aan wisselspanning gekoppeld, zodat aan de absolute polariteit niet al te zwaar getild hoeft te worden.

adres   ØF83H

Dit stuk van de toets-decoder verwerkt toetsen met nummers tussen ØØH en 2FH. De toestand van de "SHIFT", "GRPH" en "CODE"-toetsen wordt bepaald via rij 6 van NEWKEY en gecombineerd met het nummer van de toets, zodat een opzoek-adres in de tabel vanaf ØDA5H verkregen wordt. Dan wordt de toets-code uit de tabel gelezen. Is die code nul, dan stopt de routine. Is de code FFH, dan wordt verdergegaan met de dode-toets-verwerker (ØF1FH).

Ligt de code tussen 4ØH en 5FH of tussen 6ØH en 7FH, en de "CTRL"-toets werd ingedrukt, dan wordt de overeenkomstige toets-code in KEYBUF gezet (ØF55H).

Ligt de code tussen Ø1H en 1FH, dan wordt eerst een grafische header-code (Ø1H) in KEYBUF gezet (ØF55H), gevolgd door de eigenlijke code, waarbij 4ØH werd opgeteld.

Ligt de code tussen 61H en 7BH, en CAPST geeft aan dat de hoofdletters ingeschakeld zijn, dan wordt de code omgezet in een hoofdlettercode door er 2ØH van af te trekken. In de veronderstelling dat KANAST nul bevat (altijd, bij een Engelse machine), wordt de code in KEYBUF gezet en de routine stopt. Bij Europese machines, met een dode toets in plaats van een Pond-toets, kunnen de codes die met een klinker overeenkomen dan nog verder in grafische codes omgezet worden.

adres   ØFC3H

Dit stuk van de toets-decoder verwerkt de vijf functietoetsen. De toestand van "SHIFT" wordt onderzocht via rij 6 van NEWKEY, en er wordt vijf opgeteld bij het nummer van de toets, indien "SHIFT" ingedrukt is. Dan wordt de controle overgedragen naar ØEC5H, waar verder verwerkt wordt.

adres   1Ø21H

Deze routine doorzoekt de tabel vanaf adres 1B97H, om te bepalen bij welke groep toetsen het toets-nummer in register C hoort. Het overeenkomstige adres wordt uit de tabel gehaald en de controle wordt overgedragen aan dat deel van de toets-decoder. Merk op dat de eigenlijke tabel in feite midden in de standaardroutine OUTDO staat; dat is het gevolg van wijzigingen aan de Japanse ROM.

adres 1033H

In deze tabel staan de toets-codes van de toetsen met nummers 30H tot 57H, behalve de toetsen "CAP", "F1" tot "F5", "STOP" en "HOME". Indien in de tabel nul staat, wordt geen toetscode geproduceerd bij het indrukken van die toets.

00H	00H	00H	00H	00H	00H	00H	00H	RIJ 6
0DH	18H	08H	00H	09H	1BH	00H	00H	RIJ 7
1CH	1FH	1EH	1DH	7FH	12H	0CH	20H	RIJ 8
34H	33H	32H	31H	30H	00H	00H	00H	RIJ 9
2EH	2CH	2DH	39H	38H	37H	36H	35H	RIJ 10
7	6	5	4	3	2	1	0	KOLOM

adres 105BH

Deze routine zet KANAST op nul en gaat door naar 10C2H.

adres 1061H

Op dit adres begint een tabel met de grafische karakters die op Europese machines de klinkers a,e,i,o,u vervangen.

adres 10C2H

Deze routine telt één op bij de pointer van de toetsenbord-buffer, GETPNT of PUTPNT, die in HL aangebracht wordt. Als de pointer daardoor hoger wordt dan het einde van de buffer, dan telt hij door naar het begin.

adres 10CBH
----

naam CHGET
in n.v.t.
uit A = karakter dat werd ingetypt
wijzigt AF,EI

Deze standaardroutine haalt een karakter uit de toetsenbord-buffer. Eerst wordt gecontroleerd of er al een karakter in de buffer staat (0DA6H). Zoniet, wordt de cursor aangezet (09DAH), de buffer herhaaldelijk onderzocht tot er een karakter in voorkomt (0D6AH) en dan wordt de cursor weer uitgezet (0A27H). Het karakter wordt uit de buffer gehaald met behulp van GETPNT, die nadien opgehoogd wordt (10C2H).

adres 10F9H
----

naam CKCNTC
in n.v.t.
uit n.v.t.
wijzigt AF,EI

Deze standaardroutine controleert of de "CTRL/STOP"- of de "STOP"-toets werd ingedrukt. De BASIC Interpreter gebruikt ze tijdens het verwerken van statements die de processor druk bezig houden, zoals "WAIT" en "CIRCLE", om te zien of een programma beëindigd dient te worden. Registerpaar HL wordt nul gemaakt, en dan wordt de controle overgedragen aan de standaardroutine ISCNTC.

Wanneer de Interpreter aan het werk is, bevat registerpaar HL normaal het adres van het op dat ogenblik verwerkte karakter in de BASIC programmaregel. Wordt de standaardroutine ISCNTC door een "CTRL/STOP" beëindigd, dan wordt dit adres in OLDTXT gezet door de "STOP"-statement verwerker (63E3H) waar een eventueel "CONT"-commando het later kan ophalen. Wanneer registerpaar HL eerst nul wordt gemaakt, wordt daardoor voor de "CONT"-verwerker duidelijk dat de "STOP" optrad in de loop van de afhandeling van een statement. Als antwoord op het eerstvolgende "CONT"-commando zal de foutmelding "Can't CONTINUE" worden gegeven.

adres 1102H
----

naam WRTPSG
in A = registernummer, E = databyte
uit n.v.t.
wijzigt EI

Deze standaardroutine schrijft één byte naar één van de zestien registers van de PSG. Het nummer van het register wordt naar de Adrespoort van de PSG geschreven, en het databyte naar de Datapoort.

adres 110EH
----

naam RDPSG
in A = registernummer
uit A = uit de PSG gelezen byte
wijzigt A

Deze standaardroutine leest een byte uit één van de zestien registers van de PSG. Het nummer van het register wordt naar de Adrespoort van de PSG geschreven, en het databyte gelezen uit de Datapoort.

adres 1113H
----

naam BEEP
in n.v.t.
uit n.v.t.
wijzigt AF,BC,E,EI

Deze standaardroutine laat de PSG een piepgeluid produceren. Kanaal A wordt klaargezet om een toon van 1316 Hz voort te brengen en wordt dan in werking gesteld met een amplitude van zeven. Na 49 msec wordt de controle overgedragen aan GICINI, waar de PSG opnieuw wordt geïnitialiseerd.

adres 113BH

De interrupt-verwerker gebruikt deze routine om een muziek-rij af te handelen. Omdat er drie rijen zijn, die elk een kanaal van de PSG bedienen, wordt in register A gespecificeerd welke rij moet worden afgehandeld : 0 = VOICAQ, 1 = VOICBQ, 2 = VOICCQ.

Elke string in een "PLAY"-statement wordt door de BASIC Interpreter vertaald in een reeks data-pakketten. Die worden in de toepasselijke rij geplaatst, gevolgd door een eindbyte (FFH). Hoe de pakketten moeten verwerkt worden, gedecodeerd en naar de

PSG toe vertaald, wordt aan de interrupt-verwerker overgelaten. Op die manier kan de Interpreter dus onmiddellijk met het volgende statement doorgaan, zonder op het einde van een bepaalde noot te moeten wachten.

De eerste twee bytes van een data-pakket bepalen hoeveel bytes er komen en de duur van het pakket. De hoogste drie bits van het eerste byte geven aan hoeveel bytes er in het pakket na de header komen. De rest van de header bevat de tijdsduur van het "event" in eenheden van 20 msec. Die teller geeft aan hoe lang het duurt vooraleer het volgende pakket uit de rij gelezen zal worden.

7	6	5	4	3	2	1	0
aantal bytes			tijdsduur (MSB)				
tijdsduur (LSB)							

Fig. 37 : header van een data-pakket

Deze header kan gevolgd worden door een aantal blokken, in een willekeurige volgorde, die informatie bevatten over de frekwentie of de amplitude :

7	6	5	4	3	2	1	0
0	0	x	x	frekwentie(MSB)			
frekwentie (LSB)							

Frekwentie-blok

7	6	5	4	3	2	1	0
x	1	x	x	x	x	x	x
envelope-frekwentie (MSB)							
envelope-frekwentie (LSB)							

Envelope-blok

7	6	5	4	3	2	1	0
1	x	x	mode	amplitude/vorm			

Amplitude-blok

Fig. 38 : types van informatieblokken voor een data-pakket

Vooraf bepaalt de routine waar de tijdsduur-teller staat in de overeenkomstige kanaal-buffer (VCBA, VCBB, VCBC) via GETVCP, en trekt daar 1 van af. Wordt daardoor de teller nul, dan is het tijd om het volgende pakket uit de rij te halen; zoniet, stopt

de routine.
Het nummer van de rij wordt in QUEUEN geplaatst, en uit de rij wordt een byte gelezen (11E2H). Is dat FFH, dan is het einde van de rij bereikt, en de routine stopt (11B0H). Zoniet wordt het aantal bytes in register C gezet, en het MSB van de tijdsduur in de overeenkomstige kanaalbuffer. Het tweede byte wordt gelezen (11E2H) en het LSB van de tijdsduur wordt in de overeenkomstige kanaalbuffer gezet. Aan de hand van het aantal bytes wordt bepaald of er nog meer na de header komt . Komt er niets meer, dan stopt de routine. Komt er nog een pakket, dan worden de opeenvolgende bytes uit de rij gelezen, en overeenkomstig gehandeld, tot het vooraf bepaalde aantal bytes is gelezen.

Wordt een frekwentie-blok gevonden, dan wordt een tweede byte gelezen en beide bytes worden naar registers 0 en 1, 2 en 3, 4 en 5 van de PSG geschreven, afhankelijk van het nummer van de rij.

Indien een amplitude-blok werd gevonden, dan worden de amplitude- en mode-bits naar registers 8, 9 of 10 van de PSG geschreven, afhankelijk van het nummer van de rij. Is het modebit 1, wat betekent dat de amplitude gemoduleerd moet worden, dan wordt het byte ook naar register 13 van de PSG geschreven, om de vorm van de envelope te bepalen.

Wordt er een envelope-blok gevonden, of indien bit 6 van een amplitude-blok 1 is, dan worden nog twee bytes uit de rij gelezen, die naar registers 11 en 12 van de PSG geschreven worden.

adres 11B0H

Deze routine wordt aangeroepen wanneer een eind-byte (FFH) in een datapakket in één van de drie muziek-rijen gevonden wordt. Naar register 8, 9 of 10 wordt een amplitude-waarde 0 geschreven, afhankelijk van het nummer van de rij, om het betreffende kanaal te sluiten. Het bit in MUSICF dat overeenkomt met dat kanaal wordt op 0 gezet, en de controle wordt overgedragen aan de standaardroutine STRTMS.

adres 11C4H
----

naam	STRTMS
in	n.v.t.
uit	n.v.t.
wijzigt	AF, HL

Deze standaardroutine wordt gebruikt door de "PLAY"-statement verwerker, om een begin te maken met het decoderen van de muziekrijen door de interrupt-verwerker. Vooraf wordt MUSICF bekeken, en indien daaruit blijkt dat er al kanalen werkzaam zijn, stopt de routine zonder meer. Was dat niet zo, dan wordt PLYCNT met één verlaagd, en indien er geen "PLAY"-strings meer in de rij staan, stopt de routine. Staan er nog wel, dan worden de drie tijds-tellers in VCBA, VCBB, VCBC op 0001h gesteld, zodat bij de eerstvolgende interrupt het eerste pakket van de nieuwe groep gedecodeerd zal worden, en MUSICF wordt op 07H gezet, om alle drie de kanalen in werking te stellen.

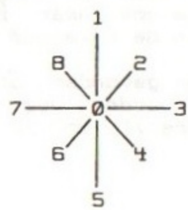
adres 11E2H

Deze routine laadt register A met het nummer van de momenteel uit te lezen rij, uit QUEUEN, en leest vervolgens een byte uit die rij (14ADH).

adres 11EEH

naam GTSTICK
in A - identificator van joystick (0, 1 of 2)
uit A - code voor de stand van de joystick
wijzigt AF, B, DE, HL, EI

Deze standaardroutine leest de stand uit van een joystick of van de vier cursor-toetsen. Als de identificator 0 is, wordt de toestand van de cursortoetsen uitgelezen via poort B van de PPI (1226H) en omgezet naar een positie-code, die opgezocht wordt in de tabel vanaf 1243H. Bij een andere identificator wordt de aansluiting 1 of 2 uitgelezen (120CH) en de vier richtings-bits omgezet in een positie-code, die opgezocht wordt in de tabel vanaf 1233H. Deze codes zijn de volgende :



adres 120CH

Deze routine leest de aansluiting van de joystick uit, die door register A bepaald wordt : 0 - aansluiting 1; 1 - aansluiting 2. De momentele waarde van register 15 van de PSG wordt uitgelezen, en dan teruggeschreven, met het "joystick-select"-bit op de juiste manier gezet. Vervolgens wordt de inhoud van register 14 van de PSG in register A gezet (11C0H), en de routine stopt.

adres 1226H

Deze routine tast rij 8 van de toetsenbord-matrix af. De momentele waarde van poort C van de PPI wordt ingelezen, en dan teruggeschreven met de vier bits die de rij op het toetsenbord aangeven, zo gewijzigd dat rij 8 aangegeven wordt. Vervolgens worden de kolomen in register A gelezen via poort B van de PPI.

adres 1253H

naam GTTRIG
in A - identificator van trigger (0,1,2,3 of 4)
uit A - status-code
wijzigt AF, BC, EI

Deze standaardroutine controleert de stand van de vuurknop op een joystick of van de "SPACE"-toets. Is de identificator 0, dan wordt rij 8 van de toetsenbord-matrix afgetast (1226H) en het resultaat omgezet in een statuscode. In de andere gevallen wordt

joystick-aansluiting 1 of 2 uitgelezen (120CH) en het resultaat daarvan in een statuscode omgezet. De identificatoren zijn :

0 - spatietoets
1 - joystick 1, vuurtoets A
2 - joystick 2, vuurtoets A
3 - joystick 1, vuurtoets B
4 - joystick 2, vuurtoets B

Wordt een bepaalde vuurtoets ingedrukt, dan is het resultaat van de aftasting FFH, zoniet 00H.

```
adres    1273H
--------------
naam     GTPDL
in       A - identificator van paddle (1 tot 12)
uit      A - waarde voor die paddle (0 tot 255)
wijzigt  AF, BC, DE, EI
```

Deze standaardroutine leest de waarde van elke paddle die aan een joystickaansluiting is aangesloten. Aan elk van de zes input-lijnen (vier voor de richting en twee vuurtoetsen) van elke aansluiting kan een paddle aangesloten worden, zodat er in totaal twaalf mogelijk zijn. De paddles aan aansluiting 1 worden geïdentificeerd als 1,3,5,7,9 en 11. De paddles aan aansluiting 2 zijn 2,4,6,8,10 en 12. Elke paddle is in de grond een monostabiele puls-generator, waarvan de pulslengte door een regelbare weerstand gestuurd wordt. Via register 15 van de PSG wordt een start-puls gestuurd naar de opgegeven joystick-aansluiting. Vervolgens wordt geteld hoe vaak register 14 van de PSG uitgelezen moet worden tot die bepaalde input terug nul wordt. Elke eenheid komt op een MSX-machine met één "WAIT"-toestand, overeen met ongeveer 12 µsec.

```
adres    12ACH
--------------
naam     GTPAD
in       A - functiecode (0 tot 7)
uit      A - status of waarde
wijzigt  AF, BC, DE, HL, EI
```

Via deze standaardroutine kan een electronisch tekenbord ("touchpad") uitgelezen worden, dat aan een van de joystick-aansluitingen is gekoppeld. De beschikbare functiecodes voor aansluiting 1 zijn :

0 - geef status (aan/uit)
1 - geef X-coördinaat
2 - geef Y-coördinaat
3 - geef stand van schakelaar

De codes 4 tot 7 leveren hetzelfde op, maar van aansluiting 2. Code 0 (4) geeft FFH als resultaat indien het bord aangeraakt wordt, zoniet 0. Code 3 (7) geeft FFH indien de schakelaar ingedrukt wordt, zoniet 0. De codes 1 en 2 (5 en 6) leveren de coördinaten op van het laatst aangeraakte punt. Die coördinaten worden opgeslagen in het werkgebied, in de variabelen PADX en PADY, wanneer middels codes 0 of 4 activiteit ontdekt wordt. Merk op dat deze variabelen door beide joystick-aansluitingen worden gebruikt.

```
adres     1384H
--------------
naam      STMOTR
in        A - code voor motor aan/uit
uit       n.v.t.
wijzigt   AF
```

Deze standaardroutine zet het relais dat de cassettemotor stuurt, aan of uit via poort C van de PPI : 00H - uit, 01H - aan, FFH - omgekeerde van de huidige stand.

```
adres     1398H
--------------
naam      NMI
in        n.v.t.
uit       n.v.t.
wijzigt   niets
```

Deze standaardroutine verwerkt normaal een niet-maskeerbare interrupt van de Z80. Op een standaard MSX-machine doet ze niets.

```
adres     139DH
--------------
naam      INIFNK
in        n.v.t.
uit       n.v.t.
wijzigt   BC, DE, HL
```

Deze standaardroutine initialiseert de strings die bij de tien functietoetsen horen, met de inhoud die ze hebben bij het aanzetten van de machine. De honderdzestig bytes vanaf 13A9H worden naar de FNKSTR-buffer in het Werkgebied gecopieerd.

```
adres     13A9H     TABEL
-------------------------
```

Op dit adres begint de tabel van 160 bytes lang, die de oorspronkelijke tekst bevat voor de functietoetsen. Elke string is 16 tekens lang, en op ongebruikte posities staan nullen :

F1 tot F5	F6 tot F10
color	color 15,4,4 + "CR"
auto	cload"
goto	cont + "CR"
list	list . + "CR" + 2 x "omhoog"-code
run + "CR"	"CLS" + run + "CR"

```
adres     1449H
--------------
naam      RDVDP
in        n.v.t.
uit       A - inhoud van statusregister van de V.D.P.
wijzigt   A
```

Deze standaardroutine leest de inhoud van het statusregister van de Video Display Processor uit via de commando-poort. Merk wel op dat deze uitlezing de bijbehorende vlaggen allemaal reset, en ook de interrupt-verwerker kan beïnvloeden.

```
adres    144CH
--------------
naam     RSLREG
in       n.v.t.
uit      A = inhoud van primair slot-register
wijzigt  A
```

Deze standaardroutine leest de inhoud van het primair slotregister via poort A van de Parallelle periferie-interface.

```
adres    144FH
--------------
naam     WSLREG
in       A = te schrijven byte
uit      n.v.t.
wijzigt  niets
```

Deze standaardroutine schrijft, via poort A van de parallelle periferie-interface, een bepaalde waarde naar het primaire slotregister.

```
adres    1452H
--------------
naam     SNSMAT
in       A = nummer van een rij van het toetsenbord
uit      A = kolom-gegevens van die rij
wijzigt  AF,C,EI
```

Deze routine leest een volledige rij van de toetsenbord-matrix uit. Poort C van de Parallelle periferie-interface wordt gelezen, en teruggeschreven, met nu op de plaats van de vier bits die de rij bepalen, het nummer van de te lezen rij. Dan wordt poort B van de Parallelle periferie-interface in register A ingelezen, waardoor de toestand van de acht kolommen bekend is. De vier andere controle-outputs van poort C van de Parallelle periferie-interface worden door deze routine niet gewijzigd.

```
adres    145FH
--------------
naam     ISFLIO
in       n.v.t.
uit      vlag NZ indien een In/Uit-file actief is
wijzigt  AF
```

Deze standaardroutine controleert of de BASIC Interpreter op dat ogenblik input of output via een in/uit-buffer leidt. Daartoe wordt PTRFIL bekeken. Normaal is die 0, maar wanneer statements zoals "PRINT#1", "INPUT#1" en dergelijke door de Interpreter verwerkt worden, staat daar een bufferadres van een controleblok voor dat bepaald kanaal (#).

```
adres    146AH
--------------
naam     DCOMPR
in       HL, DE
uit      NC als HL>DE, Z als HL=DE, C als HL<DE
wijzigt  AF
```

De BASIC Interpreter gebruikt deze standaardroutine om de waarden die in HL en DE staan, te vergelijken.

```
adres    1407H
--------------
naam     GETVCP
in       A = kanaal-nummer
uit      HL = adres in kanaalbuffer
wijzigt  AF,HL
```

Deze standaardroutine berekent het adres van byte 2 in de opgegeven kanaalbuffer (VCBA, VCBB of VCBC).

```
adres    1474H
--------------
naam     GETVC2
in       L = nummer van een byte (Ø tot 36)
uit      HL = adres in kanaalbuffer
wijzigt  AF, HL
```

Deze standaardroutine berekent het adres van een opgegeven byte in de kanaalbuffer (VCBA, VCBB of VCBC), aangegeven door het kanaalnummer in VOICEN.

```
adres    148AH
--------------
naam     PHYDIO
in       n.v.t.
uit      n.v.t.
wijzigt  niets
```

Deze standaardroutine wordt gebruikt voor Disk BASIC, en doet niets bij eèn standaard MSX machine.

```
adres    148EH
--------------
naam     FORMAT
in       n.v.t.
uit      n.v.t.
wijzigt  niets
```

Deze standaardroutine wordt gebruikt voor Disk BASIC, en doet niets bij een standaard MSX machine.

```
adres    1492H
--------------
naam     PUTQ
in       A = nummer van een rij, E = databyte
uit      vlag Z indien rij vol is
wijzigt  AF, BC, HL
```

Deze standaardroutine plaatst een databyte in één van de drie muziekrijen. De plaats waar een byte gehaald en gezet moet worden, wordt eerst gelezen in QUETAB (14FAH). De zet-positie wordt tijdelijk opgehoogd en vergeleken met de haal-positie. Als beide gelijk zijn, stopt de routine, aangezien de rij dan vol is. Is dat niet zo, dan wordt het adres van de rij uit QUETAB gehaald, en de zet-positie erbij opgeteld. Het databyte wordt op deze plaats in de rij gezet; de positie wordt opgehoogd, en de routine stopt. Merk op dat de muziekrijen circulair zijn : indien de zet- of haal-positie de laatste positie in de rij bereikt, telt ze door vanaf de start.

adres 14ADH

Deze routine laat de interrupt-verwerker toe om een byte uit één van de drie muziekrijen te lezen. In register A staat het nummer van de rij, en het databyte wordt in datzelfde register ingelezen, met vlag Z indien de rij leeg is. De zet- en haal-posities van de rij worden vooraf uit QUETAB gehaald (14FAH). Indien de terugzet-vlag actief is, wordt het databyte uit QUEBAK gehaald en de routine stopt (14D1H). Deze voorziening wordt in de huidige versie van de MSX ROM niet gebruikt. Vervolgens wordt de zet-positie met de haal-positie vergeleken. Zijn die gelijk, dan stopt de routine, omdat de rij dan leeg is. Zoniet wordt uit QUETAB het adres van de rij gelezen, en de haal-positie daarbij opgeteld. Uit de zo verkregen locatie wordt het databyte gelezen; de haal-positie wordt opgehoogd en de routine stopt.

adres 14DAH

Deze routine wordt door de standaardroutine GICINI gebruikt om een controleblok voor een rij in QUETAB te initialiseren. Vooraf wordt het controleblok in QUETAB gelocaliseerd (1504H) en de bytes die de zet- en haalpositie bevatten, op 0 gezet. Ook het terugzet-byte wordt 0 gemaakt. Het byte dat de lengte bevat, wordt uit register B gehaald en het adres van de rij uit registerpaar DE.

adres 14EBH

naam LFTQ
in A = nummer van een rij
uit HL = de nog vrije ruimte in die rij
wijzigt AF, BC, HL

Deze standaardroutine berekent het aantal bytes dat nog vrij is in een muziekrij. De haal- en zetposities worden uit QUETAB gehaald (14FAH) en de vrije ruimte wordt berekend door de zetpositie van de haalpositie af te trekken.

adres 14FAH

Deze routine leest de controle-parameters voor een muziekrij in QUETAB. Het nummer van de rij wordt in register A gezet. Eerst wordt het controleblok in QUETAB gelocaliseerd (1504H), dan wordt de zet-positie in register B geplaatst, de haal-positie in register C en de terugzet-vlag in register A.

adres 1504H

Deze routine zoekt in QUETAB een controleblok op van een bepaalde muziekrij. Het nummer van de rij moet in register A staan. Het adres van het controleblok wordt in registerpaar HL geleverd. Het nummer van de rij wordt met zes vermenigvuldigd (er zijn zes bytes per blok), en opgeteld bij het adres van QUETAB, dat in QUEUES staat.

adres 1510H

---

naam GRPPRT
in A = karaktercode
uit n.v.t.
wijzigt EI

Deze standaardroutine zet een karakter op het scherm in grafische- en veelkleurenmode. Functioneel doet ze hetzelfde als de standaardroutine CHPUT.

Vooraf wordt via de standaardroutine CNVCHR gecontroleerd of het om een grafisch karakter gaat. Gaat het om een grafische header (01H) dan stopt de routine zonder meer. Gaat het om een omgezet grafisch karakter, dan wordt het gedeelte waar de controlecodes worden gedecodeerd, overgeslagen. Zoniet, wordt gecontroleerd of het om een controlecode gaat. Enkel een "CR" wordt herkend (157EH), codes lager dan 20H worden genegeerd.

In de veronderstelling dat het karakter schrijfbaar is, wordt het pixelpatroon ervan uit de ROM karakterset gecopieerd naar de PATWRK-buffer (0752H) en FORCLR gecopieerd naar ATRBYT om de kleur ervan in te vullen. Uit GRPACX en GRPACY worden vervolgens de huidige grafische coördinaten gehaald, waaruit het adres van de pixel links bovenaan het karakter-veld berekend wordt via de standaardroutines SCALXY en MAPXYC.

In het acht bytes tellende patroon in PATWRK wordt één byte per keer verwerkt. Bij het begin van elk byte wordt het adres van de huidige pixel berekend via de standaardroutine FETCHC, en weggezet. Dan worden de acht bits na elkaar bekeken. Is een bit 1, dan wordt de overeenkomstige pixel geset door de standaard-routine SETC. Is het bit nul, dan gebeurt er niets. Na elk bit wordt de positie van de pixel op het scherm één punt naar rechts gezet (16ACH). Is het byte afgewerkt, of wordt de rechterrand van het scherm bereikt, dan wordt de oorspronkelijke pixel-positie teruggehaald en één rij omlaag gebracht door de stan-daardroutine TDOWNC.

Is het patroon afgewerkt, of wordt de onderkant van het scherm bereikt, dan wordt GRPACX aangepast. In grafische mode wil dat zeggen : vermeerderd met acht; in veelkleurenmode, met 32. Is GRPACX daarna hoger dan 255 (de rechterkant van het scherm) dan wordt een "CR" uitgevoerd (157EH).

adres 157EH

Deze routine verzorgt de "CR"-beweging voor de standaardroutine GRPPRT. Ze bestaat uit een combinatie van "CR" + "Line Feed". GRPACX wordt op 0 gezet, en bij GRPACY wordt 8 of 32 opgeteld, afhankelijk van de scherm-mode. Is GRPACY daardoor hoger dan 191 (de onderkant van het scherm) dan wordt deze 0 gemaakt.

Een toepassingsprogramma kan GRPACX en GRPACY rechtstreeks bewerken, ter compensatie van het beperkte aantal controle-functies die ter beschikking staan.

```
adres     1599H
--------------
naam      SCALXY
in        BC - X-coördinaat, DE - Y-coördinaat
uit       vlag NC indien aangepast
wijzigt   AF
```

Deze standaardroutine beperkt, indien nodig, grafische coördinaten. De BASIC Interpreter kan coördinaten berekenen tussen -32768 en +32767, hoewel dit ruimschoots buiten het schermbereik ligt. Deze routine wijzigt coördinaten die te groot zijn, zodanig dat ze binnen het fysisch bereikbare gebied vallen. Is X groter dan 255, dan wordt hij 255 gemaakt. Is Y groter dan 191, dan wordt hij 191. Is één van beide negatief (d.w.z. hoger dan 7FFFH), dan wordt hij 0. Tenslotte worden beide coördinaten, indien het scherm in de veelkleurenmode staat, door vier gedeeld, zoals vereist door de standaardroutine MAPXYC.

```
adres     15D9H
--------------
```

Met deze routine kan de huidige scherm-mode onderzocht worden. Wanneer het scherm in grafische mode staat, wordt de Z vlag geset.

```
adres     15DFH
--------------
naam      MAPXYC
in        BC - X-coördinaat, DE - Y-coördinaat
uit       n.v.t.
wijzigt   AF,D,HL
```

Met deze standaardroutine wordt een grafisch coördinatenpaar omgezet in een pixelpositie. De plaats in de Karaktertabel waar het byte staat, waarin de pixel zich bevindt, wordt in CLOC gezet. Het bitmasker, waardoor de pixel binnen het byte geïdentificeerd kan worden, wordt in CMASK gezet. Voor grafische mode wordt een iets verschillende omzetting gebruikt dan voor veelkleurenmode. De gelijklopende BASIC programma's zien er zo uit :

```
Grafische Mode
--------------
 10 INPUT "X,Y Coördinaten";X,Y
 20 A=(Y\8)*256+(Y AND 7)+(X AND &HF8)
 30 PRINT"ADRES=";HEX$(BASE(12)+A);"H ";
 40 RESTORE 100
 50 FOR N=0 TO (X AND 7): READ M$: NEXT N
 60 PRINT"MASK=";M$
 70 GOTO 10
 100 DATA 10000000
 110 DATA 01000000
 120 DATA 00100000
 130 DATA 00010000
 140 DATA 00001000
 150 DATA 00000100
 160 DATA 00000010
 170 DATA 00000001
```

Veelkleuren Mode

```
 10 INPUT "X,Y Coördinaten";X,Y
 20 X=X\4:Y=Y\4
 30 A=(Y\8)*256+(Y AND 7)+(X*4 AND &HF8)
 40 PRINT"ADRES=";HEX$(BASE(17)+A);"H ";
 50 IF X MOD 2=0 THEN M$="11110000" ELSE M$="00001111"
 60 PRINT"MASK=";M$
 70 GOTO 10
```

De toegestane waarden bij beide programma's zijn voor X: 0 tot 255 en voor Y: 0 tot 191. De data-regels in het programma voor de grafische mode komen overeen met de masker-tabel van acht bytes lang, op adres 160BH in de MSX-ROM. Regel 20 in het veelkleuren-programma komt overeen met de deling door vier die gebeurt in SCALXY. Die regel staat in het programma, om het coördinatensysteem voor beide programma's gelijk te doen lopen.

```
adres    1639H
--------------
naam     FETCHC
in       n.v.t.
uit      A = CMASK, HL = CLOC
wijzigt  A, HL
```

Deze standaardroutine haalt het adres op van de huidige pixel : registerpaar HL neemt de waarde uit CLOC en register A die uit CMASK.

```
adres    1640H
--------------
naam     STOREC
in       A = CMASK, HL = CLOC
uit      n.v.t.
wijzigt  niets
```

Deze standaardroutine zet in CLOC de inhoud van registerpaar HL (pixeladres) en in CMASK de inhoud van register A.

```
adres    1647H
--------------
naam     READC
in       n.v.t.
uit      A = kleurcode van huidige pixel
wijzigt  AF, EI
```

Deze standaardroutine berekent de kleur van een pixel. Vooraf wordt het adres in VRAM berekend via de standaardroutine FETCHC. Staat het scherm in grafische mode, dan wordt het byte waarnaar CLOC verwijst, uit de Karaktertabel gelezen via de standaardroutine RDVRM. De benodigde bit wordt dan, met behulp van CMASK, geïsoleerd, en gebruikt om in de overeenkomstige locatie van de Kleurentabel de bovenste of onderste vier bits te lezen.

Staat het scherm in veelkleurenmode, dan wordt het byte waarnaar CLOC verwijst, uit de Karaktertabel gelezen via de standaardroutine RDVRM. Dan worden met CMASK de hoogste of laagste vier bits geïsoleerd. In elk van beide gevallen zal een waarde berekend worden die een normale kleur voor de Video

Display Processor aangeeft, die ligt tussen nul en vijftien.

adres 1676H

---

naam SETATR
in A = kleurcode
uit Vlag C indien code illegaal was
wijzigt de vlaggen

Deze standaardroutine vult de inktkleur in voor grafisch gebruik, die door de standaardroutine SETC en NSETCX gebruikt wordt. De kleurcode, tussen nul en vijftien, wordt gewoon in ATRBYT gezet.

adres 167EH

---

naam SETC
in n.v.t.
uit n.v.t.
wijzigt AF, EI

Deze standaardroutine zet de laatst geschreven pixel in een bepaalde kleur. De kleurcode wordt uit ATRBYT gehaald. Vooraf wordt het adres van de pixel berekend met de standaardroutine FETCHC. In grafische mode wordt zowel de Karaktertabel als de Kleurentabel gewijzigd.

In veelkleurenmode wordt, middels de standaardroutine RDVRM, het byte waar CLOC naar verwijst uit de Karaktertabel gelezen. Vervolgens wordt de inhoud van ATRBYT in de hoogste of laagste vier bits gezet, volgens CMASK, en het byte wordt via de standaardroutine WRTVRM teruggeschreven.

adres 16ACH

Deze routine beweegt de scherm-positie van de laatste pixel één punt naar rechts. Ligt dit buiten de rand van het scherm, dan wordt de positie niet gewijzigd, en de routine stopt met vlag C. In grafische mode wordt CMASK eerst een bit naar rechts geschoven. Ligt de pixel dan nog binnen hetzelfde byte, dan stopt de routine hier. Verwijst CLOC naar de uiterst rechtse karaktercel (LSB = F8H tot FFH) dan stopt de routine met vlag C (175AH). Zoniet wordt CMASK op 80H gezet (de uiterst linkse pixel) en wordt er bij CLOC 8 opgeteld.

In veelkleurenmode wordt de controle overgedragen aan een afzonderlijke routine (1779H).

adres 16C5H

---

naam RIGHTC
in n.v.t.
uit n.v.t.
wijzigt AF

Deze standaardroutine verzet de schermpositie van de laatste pixel een punt naar rechts. In grafische mode wordt CMASK eerst een bit naar rechts geschoven. Ligt de pixel dan nog binnen hetzelfde byte, dan stopt de routine. Zoniet wordt CMASK op 80H gezet (de uiterst linkse pixel van een byte) en bij CLOC wordt 8

opgeteld. Let er op, dat de berekende positie onjuist is, als verder gegaan wordt dan de rechterrand van het scherm!

In veelkleurenmode wordt de controle overgenomen door een afzonderlijke routine (178BH).

adres 16D8H

Deze routine verzet de laatste pixel één positie naar links. Ligt hij daardoor buiten de linkerrand van het scherm, dan wordt het adres niet gewijzigd en stopt de routine met vlag C. In grafische mode wordt vooraf CMASK één bit naar links geschoven. Ligt de pixel dan nog binnen hetzelfde byte, dan stopt de routine hier. Als CLOC naar de uiterst linkse karaktercel verwijst (LSB = 00H tot 07H), dan stopt de routine met vlag C (175AH). Is dat niet zo, wordt CMASK op 01H gezet (uiterst rechtse pixel) en van CLOC wordt 8 afgetrokken.

In veelkleurenmode neemt een afzonderlijke routine de controle over (179CH).

adres 16EEH
----

naam LEFTC
in n.v.t.
uit n.v.t.
wijzigt AF

Deze standaardroutine verzet de schermpositie van de laatste pixel een punt naar links. In grafische mode wordt eerst CMASK één bit naar links geschoven. Ligt de pixel nog binnen hetzelfde byte, dan stopt de routine hier. Zoniet, wordt CMASK op 01H gezet (de uiterst rechtse pixel) en van CLOC wordt 8 afgetrokken. Let er op dat, indien de linkerrand van het scherm werd overschreden, de berekende positie onjuist is.

In veelkleurenmode neemt een afzonderlijke routine de controle over (17ACH).

adres 170AH
----

naam TDOWNC
in n.v.t.
uit vlag C indien buiten scherm
wijzigt AF

Deze standaardroutine verzet de laatste pixel een positie omlaag. Ligt die positie daardoor buiten het scherm, dan wordt het adres niet gewijzigd en stopt de routine met vlag C. In grafische mode wordt eerst CLOC met 1 vermeerderd. Valt het resultaat dan nog binnen een acht-byte-grens, dan stopt de routine. Als CLOC binnen de laatste regel (karakters) lag (CLOC >= 1700H) dan stopt de routine met vlag C (1759H). Zoniet wordt bij CLOC 00F8H opgeteld.

In veelkleurenmode wordt de controle overgedragen aan een afzonderlijke routine (17C6H).

adres 172AH

---

naam DOWNC
in n.v.t.
uit n.v.t.
wijzigt AF

Deze standaardroutine zet de laatste pixel één positie lager. In grafische mode wordt CLOC eerst met 1 vermeerderd. Valt het dan nog binnen een acht-byte grens, dan eindigt de routine. Zoniet wordt 00F8H bij CLOC opgeteld. Let er op dat, indien de onderkant van het scherm wordt overschreden, de berekende positie onjuist is!

In veelkleurenmode wordt de controle overgedragen aan een afzonderlijke routine (17DCH).

adres 173CH

---

naam TUPC
in n.v.t.
uit vlag C indien buiten scherm
wijzigt AF

Deze standaardroutine verzet de laatste pixel één positie omhoog. Wordt de bovenrand van het scherm overschreden, dan blijft de positie wat ze was en eindigt de routine met vlag C. In grafische mode wordt eerst 1 afgetrokken van CLOC; ligt dat dan nog binnen een acht-byte grens, dan eindigt de routine. Lag CLOC op de bovenste regel (karakters) (CLOC < 0100H) dan eindigt de routine met vlag C, zoniet wordt 00F8H van CLOC afgetrokken.

In veelkleurenmode wordt de controle overgedragen aan een afzonderlijke routine (17E3H).

adres 175DH

---

naam UPC
in n.v.t.
uit n.v.t.
wijzigt AF

Standaardroutine om de laatste pixel een positie hoger te zetten. In grafische mode wordt eerst CLOC met 1 verlaagd. Ligt het dan nog binnen een acht-byte grens, dan eindigt de routine. Zoniet wordt van CLOC 00F8H afgetrokken. Let er op dat,indien de bovenrand van het scherm wordt overschreden, de berekende positie niet juist is.

In veelkleurenmode wordt de controle overgedragen aan een afzonderlijke routine (17F8H).

adres 1779H

Deze routine is de veelkleuren-versie van de routine op 16ACH. Ze is identiek aan deze laatste, behalve dat hier CMASK vier keer naar rechts geschoven wordt en F0H wordt indien de grens van een karaktercel wordt overschreden.

adres 17BBH

Deze routine is de veelkleuren-versie van RIGHTC (16C5H). Ze is identiek aan deze laatste, behalve dat hier CMASK vier keer naar rechts geschoven wordt en F0H wordt indien de grens van een karaktercel wordt overschreden.

adres 179CH

Deze routine is de veelkleuren-versie van de routine op 16D8H. Ze is identiek aan deze laatste, behalve dat hier CMASK vier keer naar links geschoven wordt en 0FH wordt indien de grens van een karaktercel wordt overschreden.

adres 17ACH

Deze routine is de veelkleuren-versie van LEFTC. Ze is identiek aan deze laatste, behalve dat hier CMASK vier keer naar links geschoven wordt en 0FH wordt indien de grens van een karaktercel wordt overschreden.

adres 17C6H

Dit is de veelkleurenversie van TDOWNC (170AH). Ze is identiek aan deze laatste, behalve dat hier het grensadres voor de onderkant van het scherm 0500H is in paats van 1700H. In deze routine zit een foutje, waardoor ze onvoorspelbare dingen gaat doen indien MLTCGP, het startadres van de Karaktertabel, op een andere waarde dan nul, zijn normale waarde, gezet wordt. Op adres 17CEH hoort een EX DE,HL-instructie tussengevoegd te worden! Indien het startadres van de Karaktertabel verhoogd wordt, dan zal de routine reageren alsof de bodem van het scherm bereikt werd, hoewel dat niet zo is. Deze routine wordt door het "PAINT"-statement gebruikt. Het volgende programma illustreert de fout :

```
10 BASE(17)=&H1000
20 SCREEN 3
30 PSET(200,0)
40 DRAW"D180L100U180R100"
50 PAINT(150,90)
60 GOTO 60
```

adres 17DCH

Dit is de veelkleurenversie van DOWNC (172AH). Ze is identiek aan de routine voor de grafische mode.

adres 17E3H

Dit is de veelkleurenversie van TUPC (173CH). Ze is identiek aan de versie voor de grafische mode, behalve dat er ook een foutje in zit. Op adres 17EBH hoort een EX DE,HL-instructie tussengevoegd te worden!
Indien het startadres van de Karaktertabel verhoogd wordt, zal de routine reageren alsof ze nog binnen de tabel werkt, terwijl ze eigenlijk al boven de rand van het scherm werkt. Het effekt kan bekeken worden door in regel 40 van het vorige programma, het "R100"-gedeelte weg te halen.

adres 17F8H

Dit is de veelkleurenversie van UPC (175DH). Ze is identiek aan de versie voor de grafische mode.

adres 1809H
----

naam NSETCX
in HL - aantal te bewerken pixels
uit n.v.t.
wijzigt AF,BC,DE,HL,EI

Deze standaardroutine kleurt een reeks pixels, vanaf de laatste geplotte pixel, horizontaal en naar rechts toe. Ze kan nagebootst worden door de standaardroutine SETC en RIGHTC, maar dat zou de werking ervan aanzienlijk vertragen. Het aantal pixels dat in registerpaar HL aangegeven wordt, moet zo berekend worden dat de rechter-rand van het scherm niet overschreden wordt. Dit zou tot zeer vreemde effekten leiden. Deze routine behoudt de coördinaten van de laatst geplotte pixel (voordat de routine zelf begon te werken).

In grafische mode wordt eerst met behulp van CMASK berekend hoeveel pixels naar rechts nog in de huidige karaktercel liggen. In de veronderstelling dat de inhoud van registerpaar HL voldoende groot is, worden die dan allemaal geset. Het overblijvende aantal wordt door 8 gedeeld, om het aantal volledige karaktercellen te bepalen. Opeenvolgende bytes in de Karaktertabel worden vervolgens 0 gemaakt, en de overeenkomstige bytes in de Kleurentabel met de inhoud van ATRBYT gevuld, om deze cellen te "fillen". Wat dan nog overblijft van het oorspronkelijke aantal in registerpaar HL, wordt in een bitmasker omgezet, met behulp van een zeven bytes lange tabel op adres 185DH. Deze pixels worden daarna geset (186CH).

In veelkleurenmode wordt de controle overgedragen aan een afzonderlijke routine (18BBH).

adres 186CH

Deze routine set tot maximaal acht pixels binnen een karaktercel in een bepaalde kleur, in grafische mode. De kleurcode wordt in ATRBYT gehaald. registerpaar HL bevat het adres van het overeenkomstige byte in de Karaktertabel, en register A bevat een bitmasker (bv. 11100000) waarin elke 1 een bit voorstelt dat ingekleurd moet worden.

Indien ATRBYT overeenkomt met de bestaande kleur van een gesette pixel in het overeenkomstige byte in de Kleurentabel, dan wordt elk gespecificeerd bit in het byte in de Karaktertabel geset. Indien ATRBYT overeenkomt met de bestaande kleur van een geresette pixel in het overeenkomstige byte in de Kleurentabel, dan wordt elk gespecificeerd bit in het byte in de Karaktertabel gereset.

Indien ATRBYT met geen van beide bestaande kleuren in het byte in de Kleurentabel overeenkomt, dan wordt normaal elk gespecificeerd bit in het byte in de Karaktertabel geset, en de kleur van een gesette pixel in het byte in de Kleurentabel wordt gewijzigd. Indien dit evenwel tot gevolg zou hebben dat alle

bits in het byte in de Karaktertabel geset zouden worden, dan wordt elk gespecificeerd bit gereset en de kleur van een geresette pixel wordt in het byte in de Kleurentabel gewijzigd.

adres 18BBH

Dit is de veelkleurenversie van de standaardroutine NSETX. de standaardroutine SETC en RIGHTC worden zo vaak aangeroepen tot het opgegeven aantal pixels gekleurd zijn. De uitvoeringssnelheid is in veelkleurenmode niet zo belangrijk, omwille van de lagere scherm-resolutie en het daaruit volgende lagere aantal noodzakelijke bewerkingen.

adres 18C7H

```
naam     GTASPC
in       n.v.t.
uit      DE - ASPCT1, HL - ASPCT2
wijzigt  DE, HL
```

Deze standaardroutine berekent de verhoudingen bij een "CIRCLE"-statement, indien er geen werden gespecificeerd.

adres 18CFH

```
naam     PNTINI
in       A - grenskleur (0 tot 15)
uit      vlag C indien niet toegelaten kleur
wijzigt  AF
```

Deze standaardroutine bepaalt de grenskleur voor het "PAINT"-statement. In veelkleurenmode wordt de opgegeven kleur in BDRATR gezet. In grafische mode wordt BDRATR uit ATRBYT gecopieerd, omdat het niet mogelijk is in die mode een kleur te hebben voor de grenslijn die verschilt van de kleur voor het opgevulde vlak.

adres 18E4H

```
naam     SCANR
in       B - "fill"-schakelaar, DE - hoeveel overslaan
uit      DE - hoeveel nog overslaan, HL - aantal pixels
wijzigt  AF,BC,DE,HL,EI
```

Deze standaardroutine wordt door de "PAINT"-statement verwerker gebruikt om naar rechts toe te zoeken, vanaf de laatste pixel, tot de kleurcode van BDRATR wordt gevonden, of de rand van het scherm werd bereikt. De eindpositie wordt nu het adres van de laatste pixel, en de startpositie wordt in CSAVEA en CSAVEM gezet. De lengte van het doortrokken gebied wordt in registerpaar HL gemeten en in FILNAM+1 gezet. Normaal wordt het doorlopen gebied ingekleurd, maar dit kan verhinderd worden, maar alleen in grafische mode, door in register B nul te zetten bij het begin van de routine. Hoeveel pixels in de opgegeven kleur overgeslagen mogen worden, vanaf de oorspronkelijke uitgangspositie, wordt in registerpaar DE van tevoren bepaald. Deze voorziening wordt door de "PAINT"-statement verwerker gebruikt, bij het zoeken naar gaten in een horizontale grens, die normaal verhindert dat naar boven toe gezocht wordt.

adres 197AH
----
naam SCANL
in n.v.t.
uit HL = aantal pixels
wijzigt AF,BC,DE,HL,EI

Deze standaardroutine zoekt naar links toe, vanaf het adres van de laatste pixel, tot de kleurcode in BDRATR gevonden wordt of de rand van het scherm wordt bereikt. De eindpositie wordt het nieuwe adres van de laatste pixel, en de lengte van het doorlopen gebied wordt in registerpaar HL gezet. Het doorlopen gebied wordt altijd ingekleurd.

adres 19C7H

Deze routine wordt door SCANL en SCANR gebruikt, om de kleur van de laatste pixel te vergelijken met de grenskleur in BDRATR.

adres 19DDH
----
naam TAPOOF
in n.v.t.
uit n.v.t.
wijzigt EI

Deze standaardroutine zet de motor van de cassetterecorder uit, nadat er data op de cassette werden geschreven. Na 550 msec. (op een machine met één "WAIT"-toestand) gaat de routine over in de standaardroutine TAPIOF.

adres 19E9H
----
naam TAPIOF
in n.v.t.
uit n.v.t.
wijzigt EI

Deze standaardroutine zet de motor van de cassetterecorder uit, nadat er data van de band werden gelezen. De Modepoort van de Parallelle periferie-interface zorgt ervoor dat het relais geopend wordt. Let er op dat de interrupts op het einde van deze routine terug in werking worden gesteld : tijdens de overdracht van data van en naar cassette dienen die afgezet te worden,omdat de timing nauw luistert.

adres 19F1H
----
naam TAPOON
in A = bepalend voor lengte van de header
uit vlag C indien onderbroken door "CTRL/STOP"
wijzigt AF,BC,HL,DI

Deze standaardroutine zet de motor van de cassetterecorder aan, wacht 550 msec om de band de tijd te geven op snelheid te komen, en schrijft dan een "header" naar de cassette. Een header is een groep van digitaal hoge signalen, die voor elk datablok geschreven worden, om de baudrate te kunnen bepalen bij het teruglezen van dat blok.

De lengte van de header wordt bepaald door register A : 00H geeft een korte, iets anders geeft een lange header. De BASIC statements "SAVE", "CSAVE" en "BSAVE" genereren een lange header bij het begin van een file, en nadien korte headers tussen elke twee datablokken. Het aantal cycli in de header wordt ook beïnvloed door de ingestelde baudrate , om de tijdsduur ervan konstant te houden :

```
1200 baud kort ...  3840 cycli ... 1,5 seconde
1200 baud lang ... 15360 cycli ... 6,1 seconde
2400 baud kort ...  7936 cycli ... 1,6 seconde
2400 baud lang ... 31744 cycli ... 6,3 seconde
```

De motor wordt aangezet, en er wordt even gewacht. Dan wordt de inhoud van HEADER met 256 vermenigvuldigd en indien register A een van nul verschillende waarde bevat, nog eens met vier, om het nodige aantal cycli te verkrijgen. Vervolgens wordt het berekende aantal hooggaande pulsen geproduceerd, en de controle gaat over naar de standaardroutine BREAKX. Aangezien de "CTRL/STOP"-toetsen pas op het einde van de routine worden afgetast, kan ze niet halfweg worden onderbroken.

```
adres    1A19H
--------------
naam     TAPOUT
in       A - databyte
uit      vlag C indien onderbroken door "CTRL/STOP"
wijzigt  AF, B, HL
```

Deze standaardroutine schrijft één byte naar cassette. De MSX ROM maakt gebruik van de FSK-methode (Frequency Shift Keyed : gecodeerd door frekwentie-verschuiving) om informatie op een cassette te bewaren. Bij 1200 baud is die identiek aan de Kansas City Standard, die door de BBC wordt gebruikt voor het overbrengen van Basicode-programma's.

Bij 1200 baud wordt elk "0"-bit geschreven als een volledige LAAG-cyclus van 1200 Hz, en elk "1"-bit als twee volledige HOOG-cycli van 2400 Hz. Zo wordt de datasnelheid konstant gehouden, doordat de "0" en "1"-bits even lang duren. Werd voor 2400 baud gekozen, dan worden de frekwenties respectievelijk 2400 en 4800 Hz, maar de verhoudingen blijven dezelfde.

Een databyte wordt geschreven, voorafgegaan door een startbit (0) (1A50H), dan acht databits, met bit 0 eerst, en gevolgd door twee stop-bits (1) (1A40H). Bij 1200 baud duurt één byte dus nominaal 11*833 µsec = 9,2 msec. Na het schrijven van de stop-bits controleert de standaardroutine BREAKX of "CTRL/STOP" werd ingedrukt. Als voorbeeld geven we hieronder weer hoe het byte 43H naar cassette geschreven wordt :



Fig. 39 : databyte naar cassette schrijven

Het is van belang, geen te lange tijd te laten tussen twee bytes, wanneer data worden weggeschreven, omdat daardoor de kans

op fouten vergroot. Een pauze van 80 µsec tussen twee bytes, bijvoorbeeld, geeft bij teruglezen ongeveer twaalf procent fouten. Als tussen twee bytes in nogal veel moet gebeuren, dan kan het beste gebufferd worden, om de data als blokken, met een header, te verzamelen. Dat gebeurt ook bij het "SAVE"-commando.

adres 1A39H

Deze routine schrijft één LAAG-cyclus van ongeveer 816 µsec naar cassette. De lengte van elke helft van de cyclus wordt uit LOW gelezen. Nadien gaat de controle over naar de algemene routine die cycli produceert (1A50H).

adres 1A40H

Deze routine schrijft twee HOOG-cycli van ongeveer 396 µsec naar cassette. De lengte van elke helft van de cyclus wordt uit HIGH gelezen. Nadien gaat de controle over naar de algemene routine die cycli produceert (1A5DH).

adres 1A50H

Deze routine schrijft één cyclus naar cassette. De lengte van de eerste helft van de cyclus wordt in register L gezet, en die van de tweede helft in register H. De eerste lengte wordt afgeteld, en dan wordt het "Cas Out"-bit via de Modepoort van de Parallelle periferie-interface geset. Nu wordt de tweede lengte afgeteld, en het bit wordt gereset.

Bij alle MSX-machines heeft de Z80 een klokfrekwentie van 3,579545 MHz (280 nsec) met één "wait"-status tijdens de M1-cyclus. Deze routine telt 16 T-tijden als eenheid, zodat elke tel met 4,47 µsec overeenkomt. Er wordt ook, onverschillig om welke lengte het gaat, een extra 20,7 µsec vast bijgeteld.

adres 1A63H

---

naam TAPION
in n.v.t.
uit vlag C indien onderbroken door "CTRL/STOP"
wijzigt AF,BC,DE,HL,DI

Deze standaardroutine zet de motor van de cassetterecorder aan, en leest de cassette tot een header gevonden wordt; dan wordt de baudrate bepaald. Opeenvolgende cycli worden ingelezen, en de lengte van elke cyclus wordt gemeten (1B34H). Wanneer er 1.111 cycli gevonden worden, met minder dan 35 µsec verschil in lengte, dan gaat het om een header.

Nu worden de volgende 256 cycli gelezen (1B34H), en er wordt een gemiddelde van gemaakt, om de lengte van een HOOG-cyclus op die cassette te bepalen. Dit getal wordt met 1,5 vermenigvuldigd en in LOWLIM gezet : daardoor wordt de minimaal aanvaardbare lengte van een startbit (0) bepaald. De lengte van een HOOG-cyclus wordt in WINWID gezet. Daar wordt het opgehaald, om het verschil te maken tussen LAAG- en HOOG-cycli.

adres   1ABCH

---

naam   TAPIN
in   n.v.t.
uit   A = gelezen byte; vlag C indien CTRL/STOP of I/O Error
wijzigt AF,BC,DE,L

Deze standaardroutine leest een databyte van cassette. Eerst wordt de cassette continu gelezen, tot een startbit gevonden wordt. Dit gebeurt door te zoeken naar een laaggaand signaal, de lengte van de volgende cyclus te meten (1B1FH) en die te vergelijken met LOWLIM, om te zien of ze groter is.

De acht databits worden dan ingelezen, door het aantal hoog/laag-overgangen te tellen binnen een bepaalde periode (1B03H). Is dat aantal 0 of 1, dan gaat het om een "0"-bit. Zijn er twee of drie overgangen, dan gaat het om een "1"-bit. Worden er meer dan drie overgangen geteld, dan stopt de routine, met vlag C geset; er wordt aangenomen dat het in dat geval om een of andere fout in de hardware gaat. Na de bepaling van de waarde van elk bit, worden er nog een of twee overgangen gelezen, om de synchronisatie te behouden. Werd er voordien een oneven aantal geteld, dan wordt er nog één gelezen, in het andere geval nog twee.

adres   1B03H

Deze routine wordt door de standaardroutine TAPIN gebruikt om het aantal hoog/laag- overgangen te tellen binnen een bepaalde periode. De lengte van die periode staat in WINWID en is ongeveer 1,5 keer een HOOG-cyclus, zoals de figuur hieronder toont :

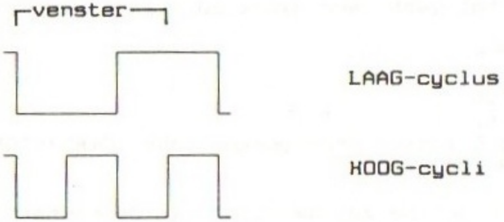


Fig. 40 : meetperiode voor overgangen

Het "Cas Input"-bit wordt voortdurend via register 14 van de Programmeerbare geluidsgenerator bekeken, en vergeleken met de voorgaande uitlezing, in register E. Elke keer als een overgang vastgesteld wordt, verhoogt register C met 1. Die controle gebeurt om de 17,3 µsec. Dat wil zeggen dat de waarde in WINWID, die door de standaardroutine TAPION werd vastgesteld, met een tel-periode van 11,45 µsec, inderdaad met 1,5 vermenigvuldigd wordt.

adres 1B1FH

Deze routine meet hoelang het duurt tot aan de volgende overgang van een cassette-input. Het "Cas Input"-bit wordt voortdurend gecontroleerd, via register 14 van de Programmeerbare geluidsgenerator, tot het een andere waarde krijgt dan in register E wordt aangegeven. Dan wordt de overeenkomstige vlag omgekeerd. De tijdsduur staat in register C, waarbij elke eenheid 11,45 μsec voorstelt.

adres 1B34H

Deze routine meet de lengte van een complete cyclus op cassette, tussen twee hoog/laag-overgangen. Via register 14 van de Programmeerbare geluidsgenerator wordt het "Cas Input"-bit bekeken, tot het nul wordt. De overgangsvlag in register E wordt op Ø gezet, en de tijd tot aan de volgende laag/hoog-overgang gemeten (1B23H). Vervolgens wordt de tijd gemeten tot aan de volgende hoog/laag-overgang (1B25H), en het totaal komt in register C.

adres 1B45H
----
naam OUTDO
in A - karakter dat verwerkt moet worden
uit n.v.t.
wijzigt EI

Deze standaardroutine wordt door de BASIC Interpreter gebruikt om een karakter naar het op dat moment actieve randapparaat te sturen. Vooraf wordt met de standaardroutine ISFLIO gecontroleerd of de output naar een I/O-buffer wordt geleid. Is dat zo, dan gaat de controle over naar de sekwentiële output-verwerker (6C48H), via de standaardroutine CALBAS. Indien PRTFLG nul is, wordt de controle overgedragen aan de standaardroutine CHPUT, om het karakter op het scherm te zetten. Als de printer actief is, wordt RAWPRT bekeken. Staat op die plaats een van nul verschillende waarde, dan wordt het karakter direkt naar de printer gestuurd (1BABH), zoniet wordt verdergegaan met de standaardroutine OUTDLP.

adres 1B63H
----
naam OUTDLP
in A - karakter dat verwerkt moet worden
uit n.v.t.
wijzigt EI

Deze standaardroutine stuurt een karakter naar de printer. Is het karakter een "TAB"-code (09H) dan worden spaties gestuurd tot LPTPOS een veelvoud van 8 is (Ø,8,16, enz.). Is het karakter een "CR"-code (ØDH) dan wordt LPTPOS nul gemaakt. Een andere controlecode laat LPTPOS ongemoeid. Was het karakter een gewoon schrijfbaar karakter, dan wordt LPTPOS opgehoogd.

Als NTMSXP nul is - wat betekent dat het een printer met MSX-specificaties is - dan wordt het karakter zonder meer aan de printer doorgegeven (1BABH). Indien een gewone printer aangesloten is, wordt via de standaardroutine CNVCHR naar grafische karakters gezocht. Is het karakter een header (Ø1H)

dan doet de routine verder niets meer. Is het een omgezet grafisch karakter, dan wordt dit door een spatie vervangen. Alle andere karakters worden naar de printer gestuurd (1BACH).

adres 1B97H TABEL

Deze tabel van twintig bytes wordt door de toetsenbord-decoder gebruikt om bij een bepaald toets-nummer de bijpassende routine op te zoeken.

TOETSNUMMER	ADRES	FUNCTIE
00H tot 2FH	0F83H	Rijen 0 tot 5
30H tot 32H	0F10H	SHIFT,CTRL,GRAPH
33H	0F36H	CAP
34H	0F10H	CODE
35H tot 39H	0FC3H	F1 tot F5
3AH tot 3BH	0F10H	ESC,TAB
3CH	0F46H	STOP
3DH tot 40H	0F10H	BS,CR,SEL,SPACE
41H	0F06H	HOME
42H tot 57H	0F10H	INS,DEL,CURSOR

adres 1BABH

Deze routine wordt door de standaardroutine OUTDLP gebruikt om een karakter naar de printer te sturen. Dat gebeurt via de standaardroutine LPTOUT. Als na LPTOUT vlag C geset is, wordt de controle overgedragen aan de "Device I/O error"-generator (73B2H) via de standaardroutine CALBAS.

adres 1BBFH TABEL

De volgende 2 K bevat de karakterset die de computer gebruikt bij het aanzetten. De eerste acht bytes bevatten het bitpatroon voor karaktercode 00H, de volgende acht het patroon voor karaktercode 01H en zo verder tot en met karaktercode FFH.

adres 23BFH

---

naam PINLIN
in n.v.t.
uit HL=start van tekst, vlag C indien "CTRL/STOP"
wijzigt AF,BC,DE,HL,EI

Deze standaardroutine wordt door de Hoofdlus van de BASIC Interpreter gebruikt om een logische tekstregel die werd ingetypt, te lezen. De controle gaat over naar de standaardroutine INLIN na het punt waar de vorige lijn werd afgebroken (23E0H).

adres 23CCH

---

naam QINLIN
in n.v.t.
uit HL=start van tekst, vlag C indien "CTRL/STOP"
wijzigt AF,BC,DE,HL,EI

Deze standaardroutine wordt door het "INPUT"-statement gebruikt, om een logische tekstregel van het scherm te lezen. De karakters

"? " worden via de standaardroutine OUTDO op het scherm gezet, en de routine loopt door in de standaardroutine INLIN.

adres 23D5H

----

naam INLIN
in n.v.t.
uit HL=start van tekst, vlag C indien "CTRL/STOP"
wijzigt AF,BC,DE,HL,EI

Deze standaardroutine wordt door het "LINE INPUT"-statement gebruikt om een logische tekstregel van het scherm te lezen. De karakters worden van het toetsenbord uitgelezen tot ofwel "CR" ofwel "CTRL/STOP" wordt ingedrukt. Vervolgens wordt de logische regel karakter per karakter van het scherm gelezen, en in de tekstbuffer BUF in het Werkgebied gezet.

De laatste schermcoördinaten worden vooraf uit CSRX en CSRY gehaald en in FSTPOS gezet. Het byte in LINTTB dat overeenkomt met de regel boven de gelezen regel, wordt eerst niet-nul gemaakt (0C29H) om te voorkomen dat die regel logisch doorloopt tot op de uitgelezen regel.
Elk karakter dat wordt ingetypt en dat via de standaardroutine CHGET wordt uitgelezen, wordt (0919H) vergeleken met de tabel met edit-functies op 2439H. Dan wordt de controle aan een van de bijhorende routines overgedragen, ofwel aan de algemene verwerkingsroutine (23FFH), al naargelang. Dit duurt net zolang tot vlag C geset wordt (door "CTRL/STOP" of "CR"). Dan wordt registerpaar HL geladen met het startadres van BUF en de routine stopt. Merk op dat vlag C gereset wordt, indien vlag Z gereset is : daarmee wordt een onderscheid gemaakt tussen een onderbreking door "CR" of een beveiligde "CTRL/STOP" en een gewone "CTRL/STOP".

adres 23FFH

Deze routine verwerkt alle karakters voor de standaardroutine INLIN, op de editing-functies na. Is het karakter een "TAB"-code (09H) dan worden spaties gestuurd (23FFH) tot CSRX een veelvoud van acht plus 1 is (kolommen 1,9,17,25,33). Wanneer het om een grafische header gaat, wordt die gewoon doorgegeven aan de standaardroutine OUTDO. Andere controlecodes onder 20H worden doorgegeven, waarna INSFLG en CSTYLE op nul worden gezet. Gaat het om een schrijfbaar karakter, dan wordt eerst INSFLG gecontroleerd, en indien nodig wordt een spatie tussengevoegd (24F2H) voordat het karakter naar de standaardroutine OUTDO gestuurd wordt.

adres 2439H TABEL

Deze tabel bevat de edit-codes, die door de standaardroutine INLIN herkend worden, gekoppeld aan het adres van een routine die ze verwerkt :

CODE NAAR FUNCTIE

08H	2561H	BS, 1 positie terug
12H	24E5H	INS, schakel INSERT in of uit
1BH	23FEH	ESC, doet niets
02H	260EH	CTRL-B, naar einde van vorige woord
06H	25F8H	CTRL-F, naar begin van volgend woord
0EH	25D7H	CTRL-N, naar eind van logische regel
05H	25B9H	CTRL-E, wis tot eind van de regel
03H	24C5H	CTRL-STOP, stop uitvoering
0DH	245AH	CR, stop uitvoering
15H	25AEH	CTRL-U, wis regel
7FH	2550H	DEL, wis karakter

adres 245AH

Deze routine voert de "CR"-bewerking uit voor de standaardroutine INLIN. De startcoördinaten van de logische regel worden opgezocht (266CH) en de cursor van het scherm gehaald (0A2EH). Tot maximaal 254 karakters worden dan uit VRAM gelezen (0BD8H) en in BUF gezet. Eventuele nul-codes (00H) worden genegeerd, en alle karakters met een code lager dan 20H worden vervangen door de combinatie van een grafische header (01H) en de code + 40H. Op het einde van elke schermregel wordt in LINTTB gekeken (0C1DH) of de logische regel verder gaat op de volgende schermregel. Eventuele achteropkomende spaties worden uit BUF gehaald, en op het einde wordt een nul gezet, als merkteken. De cursor wordt terug op het scherm gezet (09E1H) en de cursorcoördinaten worden ingevuld met de waarde voor de laatste schermregel die deel uitmaakt van de logische regel; dit gebeurt via de standaardroutine POSIT. Vervolgens wordt een "LF" naar de standaardroutine OUTDO gestuurd, wordt INSFLG op nul gezet en de routine stopt met een "CR" in register A (0DH), en vlaggen NZ en C. Deze "CR"-code wordt door de hoofdlus van de standaardroutine INLIN naar het scherm gezet net voor het einde ervan.

adres 24C5H

Deze routine voert de "CTRL/STOP"-bewerking uit voor de standaardroutine INLIN. De laatste schermregel die nog bij de logische regel hoort, wordt opgezocht door in LINTTB te kijken (0C1DH); CSTYLE wordt op 0 gezet, bij het begin van BUF wordt een nul gezet en alle muziek-variabelen worden gewist via de standaardroutine GICINI. Nu wordt gekeken in TRPTBL of er een "ON STOP"-statement in werking is. Is dat zo, dan wordt de cursor gereset (24AFH) en de routine stopt met vlaggen NZ en C. BASROM wordt gecontroleerd op beveiligde ROM. Is die controle positief, dan wordt de cursor gereset (24AFH) en de routine stopt met vlaggen NZ en C. Is dat niet zo, dan wordt de cursor gereset en de routine stopt met vlaggen Z en C.

adres 24E5H

Deze routine voert de "INS"-bewerking uit voor de standaardroutine INLIN. De toestand van INSFLG op dat ogenblik wordt omgekeerd en de controle gaat over op de standaardroutine CSTYLE (242CH).

adres 24F2H

Deze routine wordt gebruikt door het stuk van de standaardroutine INLIN dat zich niet met speciale edit-toetsen ophoudt, om een spatie tussen te voegen. De cursor wordt weggehaald (0A2EH) en uit CSRX en CSRY worden de coördinaten gehaald. Het karakter dat op die plaats staat wordt uit VRAM gehaald en door een spatie vervangen (0BE6H). De opeenvolgende karakters worden dan 1 kolom naar rechts gecopieerd, tot het einde van een scherm-regel.

Op dit punt wordt LINTTB gecontroleerd (0C1DH) om te zien of de logische regel verder gaat. Is dat zo, dan gaat de bewerking op de volgende regel verder. Zoniet, wordt het karakter dat op de laatste kolom stond, bekeken. Als het een spatie was, stopt de routine nadat ze de cursor opnieuw op het scherm heeft gezet (09E1H). Was het een ander karakter, dan wordt de overeenkomstige plaats in LINTTB op 0 gezet, om aan te geven dat die logische regel verder loopt. Het nummer van de volgende schermregel wordt vergeleken met het aantal regels op het scherm (0C32H). Is die regel de laatste, dan wordt het scherm 1 regel naar boven gescrolld (0A88H). Was dat niet zo, dan wordt een blanco regel ingevoegd (0AB7H) en het copiëren gaat verder.

adres 2550H

Deze routine voert de "DEL"-bewerking uit voor de standaardroutine INLIN. Als de cursorpositie de meest rechtse kolom aangeeft, en de logische regel niet verderloopt, dan wordt enkel een spatie naar VRAM geschreven (2595H). Is dat niet zo, dan wordt een "RECHTS"-code (1CH) naar de standaardroutine OUTDO gestuurd en de controle gaat naar de "BS"-routine.

adres 2561H

Deze routine voert de "BS"-bewerking uit voor de standaardroutine INLIN. Eerst wordt de cursor weggehaald (0A2EH) en de kolomcoördinaat van de cursor met 1 verlaagd, tenzij hij uiterst links stond, en de vorige regel niet doorliep. Vervolgens worden karakters uit VRAM gelezen (0BD8H) en één positie meer naar links teruggeschreven (0BE6H) tot op het einde van de logische regel. Dan wordt een spatie naar VRAM geschreven (0BE6H) en de cursor wordt terug op het scherm gezet (09E1H).

adres 25AEH

Deze routine voert de "CTRL"-"U"-bewerking uit voor de standaardroutine INLINE. De cursor wordt weggehaald (0A2EH) en de start van de logische regel wordt opgezocht (266CH). Die coördinaten worden in CSRX en CSRY gezet, en de hele logische regel wordt gewist (25BEH).

adres 25B9H

Deze routine voert de "CTRL"-"E"-bewerking uit voor de standaardroutine INLINE. De cursor wordt weggehaald (0A2EH) en de rest van de schermregel gewist (0AEEH). Dit gebeurt zoveel keer tot het eind van de logische regel wordt bereikt; dit wordt in LINTTB gecontroleerd (0C1DH). Daarna wordt de cursor teruggezet (09E1H), INSFLG wordt op 0 gezet en in CSTYLE wordt

de code voor "blokvormige cursor" gezet (242DH).

adres 25F8H

Deze routine voert de "CTRL"-"F"-bewerking uit voor de standaardroutine INLINE. De cursor wordt weggehaald (0A2EH) en net zo vaak naar rechts bewogen (2624H) tot een niet alfanumeriek karakter gevonden wordt. Daarna wordt hij weer naar rechts verzet (2624H) tot er een alfanumeriek karakter gevonden wordt. Nu wordt de cursor terug op het scherm gezet (25CDH) en de routine stopt.

adres 260EH

Deze routine voert de "CTRL"-"B"-bewerking uit voor de standaardroutine INLINE. De cursor wordt weggehaald (0A2EH) en daarna zo vaak naar links bewogen (2634H) tot een alfanumeriek karakter gevonden wordt. Daarna wordt hij verder naar rechts geschoven tot een niet alfanumeriek karakter gevonden is, en dan 1 positie naar rechts (0A5BH). De cursor wordt weer op het scherm gezet (25CDH) en de routine stopt.

adres 2624H

Deze routine verzet de cursor één positie naar rechts (0A5BH), laadt dan register D met het nummer van de uiterst rechtse kolom en register E met het nummer van de onderste regel op het scherm, en controleert dan of op de cursor-positie een alfanumeriek karakter staat (263DH).

adres 2634H

Deze routine verzet de cursor één positie naar links (0A4CH), laadt register D met het nummer van de uiterst linkse kolom en register E met het nummer van de bovenste regel op het scherm. De coördinaten van de cursor worden met deze waarden vergeleken en de routine stopt met vlag Z indien ze gelijk zijn. In het andere geval wordt het karakter op die plaats uit VRAM gelezen (0BD8H) en er wordt gecontroleerd of het een alfanumeriek karakter is. Zo ja, worden de vlaggen NZ en C gezet; zo nee, worden de vlaggen NZ en NC gezet en de routine stopt.

Alfanumerieke karakters zijn : de cijfers "0" tot "9", de letters "A" tot "Z" en de leters "a" tot "z". Daar worden ook de grafische karakters 86H tot 9FH en A6H tot FFH bijgenomen : dat waren oorspronkelijk Japanse lettertekens. Die hadden eigenlijk uitgesloten moeten worden, toen de ROM voor Europa werd aangepast.

adres 266CH

Deze routine zoekt de start op van een logische regel, en zet de scherm-coördinaten ervan in registerpaar HL. Elke schermregel boven de laatste wordt via LINTTB gecontroleerd (0C1DH) tot een regel wordt gevonden die logisch op die schermregel eindigt. De regel die op het scherm onmiddellijk daaronder ligt, is het begin van de logische regel, en het regelnummer ervan wordt in register L gezet. Dit nummer wordt vergeleken met de inhoud van FSTPOS, waar het regelnummer in staat op het ogenblik dat de standaardroutine INLIN begon te werken, om te zien of de cursor

nog op dezelfde regel staat. Als dat zo is, wordt de kolom-coördinaat in register H op de oorspronkelijke positie (uit FSTPOS) gezet. Was het niet dezelfde regel, dan wordt register H ingevuld met de uiterst linkse positie, zodat de hele regel gelezen kan worden.

adres 2680H

Op dit adres staat een sprong-instructie naar de initialisatie-routine bij het inschakelen van de machine (7C76H).

adres 2683H

Op dit adres staat een sprong-instructie naar de standaard-routine SYNCHR (558CH).

adres 2686H

Op dit adres staat een sprong-instructie naar de standaard-routine CHRGTR (4666H).

adres 2689H

Op dit adres staat een sprong-instructie naar de standaard-routine GETYPR (5597H).

HOOFDSTUK 5

De BASIC Interpreter

Microsoft BASIC is door de jaren heen geëvolueerd tot het zijn huidige positie als industriële standaard bereikte. De taal werd oorspronkelijk voor de 8080 processor geschreven, en zelfs de versie die MSX gebruikt, is nog in 8080-Assembler. Door deze voortdurende ontwikkeling treffen we dan ook minder specifieke Z80-instructies aan dan we in een modern programma kunnen verwachten.

Ontwikkeling houdt ook in, dat er talrijke wijzigingen werden aangebracht, zodanig dat het resultaat een nogal kronkelig programma is geworden. Door de structuur van de Interpreter wordt het weinig waarschijnlijk dat een toepassingsprogramma de vele voorzieningen die erin vervat liggen, ook zou kunnen gebruiken. Toch zal er een zekere mate van samenwerking nodig zijn tussen toepassingsprogramma's en de Interpreter. Daarom geven we hieronder een gedetailleerde beschrijving van de werking.

De Interpreter bevat vier belangrijke, duidelijk identificeerbare blokken. Het blok waar gebruikers het meest mee vertrouwd zijn is de HOOFDLUS (4134H). De Hoofdlus leest genummerde regels die ingetypt worden, en zet die op de juiste plaats (numerieke volgorde) in het Programmageheugen. Dat gaat net zolang door tot een direct commando wordt gelezen.

De VERWERKINGSLUS (4601H) staat in voor de uitvoering van een programma. Hij leest de eerste commando-code in een programmaregel en voert een CALL uit naar de overeenkomstige routine, die dan de rest van het statement verwerkt. Dit gaat verder tot geen programma-code meer wordt gevonden. Op dat punt wordt de controle overgedragen aan de Hoofdlus.

Het onderzoek van numerieke uitdrukkingen of strings gebeurt via de EXPRESSION EVALUATOR (routine die uitdrukkingen als geheel bekijkt, vaststelt of ze syntactisch correct zijn en vervolgens uitwerkt) (4C64H).

Elke uitdrukking bestaat uit factoren, die op hun beurt worden geanalyseerd door de FACTOR EVALUATOR (routine die elke factor in een uitdrukking afzonderlijk evalueert) (4DC7H). Tussen die factoren staan in de tekst, operatoren die factoren twee aan twee samenvoegen.

Er bestaan verschillende types van operands, bijvoorbeeld regelnummers, die geen deel kunnen uitmaken van een uitdrukking in Microsoft BASIC. Daarom wordt verder de term "geëvalueerd" gebruikt voor operands die wèl in een uitdrukking kunnen voorkomen, en de term "berekend" voor degene die niet in uitdrukkingen zijn toegelaten.

Een belangrijk punt waar bij het uitpluizen van de Interpreter op gelet moet worden, is dat er veel trucs worden gebruikt. De programmeurs lijken een duidelijke voorliefde te hebben om midden in een instructie te springen, om meer dan één toegangsadres voor een bepaalde routine te creëren. Neem bijvoorbeeld de instructie :

```
3E D1      NORMAAL : LD A,ØD1H
```

Deze instructie zal de accumulator met de waarde D1H laden. Indien de instructie aangesproken wordt op adres "NORMAAL+1", dan wordt een D1H, een POP DE, uitgevoerd. De Interpreter bevat veel van dergelijke obscure stukken code.

adres 268CH

Deze routine wordt gebruikt door de Expression Evaluator om een aftrekking uit te voeren op twee dubbele precisie-operands. De eerste staat in DAC, de tweede in ARG. Het resultaat van de bewerking wordt in DAC gezet. Het teken van de mantisse van de tweede operand wordt omgekeerd, en de routine loopt door in de optel-routine.

adres 269AH

Deze routine wordt gebruikt door de Expression Evaluator om twee operands met dubbele precisie bij elkaar op te tellen. De eerste staat in DAC, de tweede in ARG en het resultaat komt in DAC. Als de tweede operand nul is, stopt de routine zonder meer. Als de eerste operand nul is, wordt de tweede in DAC gecopieerd (2F05H) en de routine stopt. De twee exponenten worden met elkaar vergeleken, en indien ze meer van elkaar verschillen dan 10^15, wordt het resultaat gelijk aan de grootste operand en de routine stopt. Is het verschil kleiner, dan wordt het gebruikt om de twee mantissen van gelijke orde te maken door de kleinste naar rechts te schuiven (27A3H), zoals bijvoorbeeld :

19,2100 = 0,1921*10^2 = 0,192100
+0,7436 = 0,7436*10^0 = 0,007436

Wanneer de tekens van de twee mantissen gelijk zijn, dan worden de mantissen bij elkaar opgeteld (2759H). Verschillen ze, dan worden ze afgetrokken (276BH). De exponent van het resultaat is gewoon de grootste van de twee oorspronkelijke exponenten. Wordt een "overflow" geproduceerd, dan wordt de mantisse één plaats naar rechts geschoven (27DBH) en de exponent met 1 verhoogd. Werden door de aftrekking nullen na de komma geproduceerd dan wordt de mantisse genormaliseerd door ze naar links te schuiven (2798H). Het bewakingsbyte wordt bekeken en indien het vijftiende cijfer groter dan of gelijk aan 5 is, wordt het resultaat naar boven afgerond.

adres 2759H

Deze routine telt de twee mantissen met dubbele precisie die in DAC en ARG staan, bij elkaar op en zet het resultaat in DAC. De optelling begint bij de minst belangrijke bytes , DAC+7 en ARG+7, en gebeurt voor de zeven bytes met twee cijfers tegelijk.

adres 276BH

Deze routine trekt twee mantissen met dubbele precisie die in DAC en ARG staan, van elkaar af, en zet het resultaat in DAC. De aftrekking begint bij het bewakingsbyte, DAC+8 en ARG+8, en verloopt voor de acht bytes met twee cijfers per keer. Geeft het resultaat een "underflow", dan wordt dit gecorrigeerd door het van 0 af te trekken en het teken van de mantisse om te keren. Een voorbeeld:

0,17-0,85 = 0,32 -> -0,68

adres 2797H

Deze routine schuift de mantisse van een getal met dubbele precisie, die in DAC staat, 1 cijfer naar links.

adres 27A3H

Deze routine schuift een mantisse van een getal met dubbele precisie naar rechts. Het aantal te verschuiven bits, staat in register A; het adres van de MSB van de mantisse staat in registerpaar HL. Eerst wordt het aantal posities door twee gedeeld, om het aantal bytes van het aantal cijfers te scheiden. Het nodige aantal hele bytes wordt dan naar rechts geschoven, en de MSB's nul gemaakt. Was het opgegeven aantal posities oneven, dan wordt de mantisse nog één cijfer naar rechts geschoven.

adres 27E6H

Deze routine wordt door de Expression Evaluator gebruikt om twee operands met dubbele precisie met elkaar te vermenigvuldigen. De eerste staat in DAC, de tweede in ARG. Het resultaat wordt in DAC gezet. Als één van beide 0 is, wordt het resultaat nul gemaakt en de routine stopt (2E7DH). In het andere geval worden beide exponenten bij elkaar opgeteld, wat de exponent van het resultaat oplevert. Is die kleiner dan 10^-63, dan wordt het resultaat nul. Is hij groter dan 10^63 dan wordt een "Overflow Error" gemeld (4067H). Vervolgens worden de tekens van de mantissen verwerkt, om het resultaat van de bewerking te berekenen. Zijn ze gelijk, dan is het resultaat positief, zoniet is het negatief.

Hoewel de mantissen in BCD-formaat staan, worden ze toch vermenigvuldigd met gebruik van de normale binaire methode van optellen en verschuiven. Met dit doel wordt de eerste operand een aantal keer met twee vermenigvuldigd, om de constanten X*80, X*40, X*20, X*10, X*8, X*4, X*2 en X in de HOLD8 buffer te verkrijgen. De tweede operand blijft in ARG, en DAC wordt nul gemaakt om als product-accumulator te kunnen dienen. De vermenigvuldiging gebeurt via opeenvolgende cijferparen uit de tweede operand , te beginnen met het minst belangrijke paar. Voor elk "1"-bit in het paar wordt het bijpassende veelvoud van de eerste operand bij het subtotaal bijgeteld. Bijvoorbeeld, de vermenigvuldiging 1823*96 zou geven :

1823*10010110= (1823*80)+(1823*10)+(1823*4)+(1823*2)

Na de verwerking van elk cijferpaar wordt het product twee cijfers naar rechts geschoven. Zijn alle zeven de paren verwerkt, dan wordt het product genormaliseerd en opgerond (26FAH) en de routine stopt.

De tijd die een vermenigvuldiging in beslag neemt, hangt grotendeels af van het aantal "1"-bits in de tweede operand. In het slechtste geval, als alle cijfers 7 zijn, kan het tot 11 msec duren. Het gemiddelde is 7 msec.

adres 288AH

Deze routine verdubbelt de mantisse van een getal met dubbele

precisie drie opeenvolgende keren, om de producten X*2, X*4 en X*8 te krijgen. Het adres van het LSB van de mantisse moet in registerpaar DE staan. De producten worden opgeslagen op opeenvolgende dalende adressen, te beginnen onmiddellijk onder de operand.

adres 289FH

Deze routine wordt door de Expression Evaluator gebruikt om twee operands met dubbele precisie te delen. De eerste staat in DAC, de tweede in ARG. Het resultaat komt in DAC. Is de eerste operand nul, dan stopt de routine met nul als resultaat. Is de tweede operand nul, dan wordt een "Division by zero"-error gemeld (4058H). In de andere gevallen worden de exponenten van elkaar afgetrokken, wat de exponent van het resultaat oplevert, en de twee tekens van de mantissen worden verwerkt om het teken van het resultaat te kennen. Zijn ze gelijk, dan is het resultaat positief; zoniet, is het negatief.

De deling van de mantissen gebeurt met gebruik van de normale, lange deling. De tweede operand wordt herhaaldelijk van de eerste afgetrokken tot aan "underflow", wat één cijfer van het resultaat oplevert. Dan wordt de tweede operand terug opgeteld, waardoor de rest verkregen wordt (2761H). Het cijfer wordt in HOLD8 opgeslagen en de eerste operand wordt één cijfer naar links geschoven. Wanneer die volledig "weggeschoven" is, wordt het resultaat uit HOLD8 naar DAC gecopieerd, genormaliseerd en opgerond (2883H). De tijd om een deling uit te voeren kan maximaal zo'n 25 msec bedragen, als de eerste operand hoofdzakelijk uit negenen en de tweede uit enen bestaat, omdat dit het grootste aantal aftrekkingen vergt.

adres 2993H

Deze routine wordt door de Factor Evaluator gebruikt om op een dubbele precisie-operand in DAC, de "COS"-functie toe te passen. Eerst wordt de operand met 1/(2*PI) vermenigvuldigd, zodat de eenheid overeenkomt met een volledige cirkel, 360 graden (2C3BH). Dan wordt er 0,25 van afgetrokken (dat is 90 graden) (2C32H), het teken van de mantisse wordt omgekeerd (2E8DH) en de routine loopt door in de "SIN"-routine.

adres 29ACH

Deze routine wordt door de Factor Evaluator gebruikt om op een operand met dubbele precisie in DAC, de "SIN"-functie toe te passen. Eerst wordt de operand vermenigvuldigd met 1/(2*PI) (2C3BH) zodat de eenheid overeenkomt met een volledige cirkel van 360 graden. Aangezien de functie periodisch is, moet enkel het fractionele gedeelte van de operand gekend zijn. Dit wordt berekend door de operand te PUSHen (2CCCH), het gehele gedeelte ervan te berekenen (30CFH) en dit in ARG te copiëren, vervolgens de hele operand te POPpen in DAC (2CE1H) en het geheel gedeelte ervan af te trekken (268CH).

Nu wordt het eerste cijfer van de mantisse bekeken, om het kwadrant te bepalen. Ligt het in het eerste kwadrant, dan gebeurt er niets mee. Ligt het in het tweede kwadrant, dan wordt het van 0,5 afgetrokken (180 graden) om het in de Y-as te spiegelen. Ligt het in het derde kwadrant, dan wordt het van 0,5

afgetrokken (180 graden) om het in de X-as te spiegelen. Ligt het in het vierde kwadrant, dan wordt er 1 van afgetrokken (360 graden) om het in beide assen te spiegelen. Vervolgens wordt de functie berekend door benadering met een veelterm (2C88H) waarbij de lijst met coëfficienten op 2DEFH gebruikt wordt. Dat zijn de eerste acht termen van de reeks van Taylor :

X - (X^3/3!) + (X^5/5!) - (X^7/7!)...

waarbij de coëfficienten met opeenvolgende factoren van 2*PI worden vermenigvuldigd, als compensatie voor het oorspronkelijk op schaal brengen.

adres 29FBH

Deze routine wordt door de Factor Evaluator gebruikt om de "TAN"-functie toe te passen op een operand met dubbele precisie in DAC. De berekening gebeurt met de formule : TAN(X) = SIN(X)/COS(X).

adres 2A14H

Deze routine wordt door de Factor Evaluator gebruikt om de "ATN"-functie toe te passen op een operand met dubbele precisie in DAC. De functie wordt berekend door benadering met een veelterm (2C88H), waarbij gebruik gemaakt wordt van de lijst met coëfficienten op 2E30H. Die lijst bevat de eerste acht termen in de reeks van Taylor : X - (X^3/3!) + (X^5/5!) - (X^7/7!) ... met licht gewijzigde coëfficienten, om de serie langer te maken.

adres 2A72H

Deze routine wordt door de Factor Evaluator gebruikt om de "LOG"-functie toe te passen op een operand met dubbele precisie in DAC. De berekening gebeurt door benadering met een veelterm, waarbij de lijst met coëfficienten op 2DA5H gebruikt wordt.

adres 2AFFH

Deze routine wordt door de Factor Evaluator gebruikt om de "SQR"-functie toe te passen op een operand met dubbele precisie in DAC. De berekening gebeurt met de Newton-Raphson-methode. Een BASIC-pogramma dat op dezelfde manier werkt is dit :

```
10 INPUT"NUMBER";X
20 POGING=10
30 FOR N=1 TO 7
40 POGING=(POGING+X/POGING)/2
50 NEXT N
60 PRINT POGING
70 PRINT SQR(X)
```

Het bovenstaande programma gebruikt een eerste benaderende oplossing . Dit kan accuraat zijn binnen een bepaald bereik, maar maximale juistheid kan enkel behaald worden indien die eerste benadering dichtbij de echte wortel ligt. De ROM gebruikt de volgende methode : de exponent wordt gehalveerd en opgerond; dan worden de eerste twee cijfers van de operand door vier gedeeld, en het eerste wordt met 1 verhoogd.

adres 2B4AH

Deze routine wordt door de Factor Evaluator gebruikt om de "EXP"-functie toe te passen op een operand met dubbele precisie in DAC. Eerst wordt die met 0,4342944819 vermenigvuldigd (dat is LOG(e) basis 10), zodat het probleem om e^x uit te werken nu vereenvoudigd wordt tot de uitwerking van 10^x. Dit is veel eenvoudiger, aangezien het integer gedeelte gemakkelijk verwerkt kan worden. Na deze vereenvoudiging wordt de functie berekend met gebruik van de benadering met een veelterm. Daarbij wordt de lijst met coëfficienten op 2D6BH gebruikt.

adres 2BDFH

Deze routine wordt door de Factor Evaluator gebruikt om de "RND"-functie toe te passen op een operand met dubbele precisie in DAC. Als de operand 0 is, wordt het huidige "random"-getal uit RNDX naar DAC gecopieerd, en de routine stopt. Als de operand negatief is, wordt hij in RNDX gecopieerd, om het nieuwe toevals-getal te initialiseren. Dit nieuwe toevalsgetal wordt gegenereerd door RNDX naar HOLD8 te copiëren, de constante op adres 2CF9H naar ARG, de constante op adres 2CF1H naar DAC en dan te vermenigvuldigen (282EH). De veertien minst belangrijke cijfers van het product (dubbele precisie) worden naar RNDX gecopieerd, als mantisse voor het nieuwe toevalsgetal. De exponent in DAC wordt op 10^0 gezet, wat een getal tussen 0 en 1 oplevert.

adres 2C24H

Deze routine wordt gebruikt door de routines die "NEW", "CLEAR" en "RUN" verwerken, om RNDX te initialiseren op de constante die op adres 2D01H staat.

adres 2C2CH

Deze routine telt de constante, waarvan het adres in registerpaar HL staat, op bij de operand met dubbele precisie in DAC.

adres 2C32H

Deze routine trekt de constante, waarvan het adres in registerpaar HL staat, af van de operand met dubbele precisie in DAC.

adres 2C3BH

Deze routine vermenigvuldigt de operand met dubbele precisie, die in DAC staat, met de constante waarvan het adres in registerpaar HL staat.

adres 2C41H

Deze routine deelt de operand met dubbele precisie in DAC, door de constante waarvan het adres in registerpaar HL staat.

adres 2C47H

Deze routine voert relatie-bewerkingen uit tussen de operand met dubbele precisie in DAC en de constante waarvan het adres in registerpaar HL moet staan.

adres 2C4DH

Deze routine copieert een operand met dubbele precisie van 8 bytes lang, van DAC naar ARG.

adres 2C59H

Deze routine copieert een operand met dubbele precisie van 8 bytes lang, van ARG naar DAC.

adres 2C6FH

Deze routine verwisselt de 8 bytes in DAC met de 8 bytes die het laatst op de stack van de Z80 werden gezet.

adres 2C80H

Deze routine keert het teken om van de mantisse van de operand in DAC (2E8DH). Vervolgens wordt ditzelfde adres op de stack gezet, om op het einde van de routine het teken op de oorspronkelijke waarde te zetten.

adres 2C88H

Deze routine genereert een oneven reeks, gebaseerd op de operand met dubbele precisie in DAC. De reeks heeft de vorm :

X^1*(Kn)+X^3*(Kn-1)+X^5*(Kn-2)+X^7*(Kn-3) ...

Het adres van de lijst met coëfficienten wordt in registerpaar HL aangebracht. Het eerste byte van de lijst bevat het aantal coëfficienten. Daarna volgen de coëfficienten met dubbele precisie, met eerst K1 en als laatste Kn. De even reeks wordt gegenereerd (2C9AH) en vermenigvuldigd (27E6H) met de oorspronkelijke operand.

adres 2C9AH

Deze routine genereert een even reeks, gebaseerd op de operand met dubbele precisie in DAC. De reeks heeft de vorm :

X^0*(Kn)+X^2*(Kn-1)+X^4*(Kn-2)+X^6*(Kn-3) ...

Het adres van de lijst met coëfficienten wordt in registerpaar HL aangebracht. Het eerste byte van de lijst bevat het aantal coëfficienten. Daarna volgen de coëfficienten met dubbele precisie, met eerst K1 en als laatste Kn. Om de veelterm uit te werken wordt de methode van Horner gebruikt. Daarbij is per term slechts één vermenigvuldiging en één optelling nodig. Het equivalent in BASIC is :

```
10 X=X*X
20 PRODUCT =0
30 RESTORE 100
40 READ COUNT
50 FOR N= 1 TO COUNT
60 READ K
70 PRODUCT =(PRODUCT*X)+K
80 NEXT N
90 END
100 DATA 8
110 DATA Kn-7
120 DATA Kn-6
130 DATA Kn-5
140 DATA Kn-4
150 DATA Kn-3
160 DATA Kn-2
170 DATA Kn-1
180 DATA Kn
```

De veelterm wordt verwerkt vanaf de laatste coëfficient tot en met de eerste, zodanig dat het deelproduct gebruikt kan worden. Daardoor worden onnodige bewerkingen vermeden.

adres 2CC7H

Deze routine zet een operand met dubbele precisie van acht bytes lang, die in ARG staat, op de stack van de Z80.

adres 2CCCH

Deze routine zet een operand met dubbele precisie van acht bytes lang, die in DAC staat, op de stack van de Z80.

adres 2CDCH

Deze routine haalt een operand met dubbele precisie van acht bytes lang van de stack, en zet hem in ARG.

adres 2CE1H

Deze routine haalt een operand met dubbele precisie van acht bytes lang van de stack, en zet hem in DAC.

adres 2CF1H TABEL

Deze tabel bevat de constanten met dubbele precisie die door de rekenkundige routines worden gebruikt. De eerste drie hebben nul als exponent, omdat ze in een speciale tussenvorm geschreven staan, die door de toevalsgenerator wordt gebruikt.

(zie tabel op volgende bladzijde)

ADRES	CONSTANTE		ADRES	CONSTANTE	
2CF1H	.14389820420821	RND	2DAEH	6.2503651127908	
2CF9H	.21132486540519	RND	2DB6H	-13.682370241503	
2D01H	.40649651372358	RNDX	2DBEH	8.5167319872389	
2D09H	.43429448190234	LOG(e)	2DC6H	5	LOG
2D11H	.50000000000000		2DC7H	1.0000000000000	
2D13H	.00000000000000		2DCFH	-13.210478350156	
2D1BH	1.0000000000000		2DD7H	47.925256043873	
2D23H	.25000000000000		2DDFH	-64.906682740943	
2D2BH	3.1622776601684	SQR(10)	2DE7H	29.415750172323	
2D33H	.86858896380650	2*LOG(e)	2DEFH	8	SIN
2D3BH	2.3025850929940	1/LOG(e)	2DF0H	-.69215692291809	
2D43H	1.5707963267949	PI/2	2DF8H	3.8172886385771	
2D4BH	.26794919243112	TAN(PI/12)	2E00H	-15.094499474801	
2D53H	1.7320508075689	TAN(PI/3)	2E08H	42.058689667355	
2D5BH	.52359877559830	PI/6	2E10H	-76.705859683291	
2D63H	.15915494309190	1/(2*PI)	2E18H	81.605249275513	
2D6BH	4	EXP	2E20H	-41.341702240398	
2D6CH	1.0000000000000		2E28H	6.2831853071796	
2D74H	159.37415236031		2E30H	8	ATN
2D7CH	2709.3169408516		2E31H	-.05208693904000	
2D84H	4497.6335574058		2E39H	.07530714913480	
2D8CH	3	EXP	2E41H	-.09081343224705	
2D8DH	18.312360159275		2E49H	.11110794184029	
2D95H	831.40672129371		2E51H	-.14285708554884	
2D9DH	5178.0919915162		2E59H	.19999999948967	
2DA5H	4	LOG	2E61H	-.33333333333160	
2DA6H	-.71433382153226		2E69H	1.0000000000000	

adres 2E71H

Deze routine berekent het teken van de mantisse van een operand met vlottende komma, die in DAC staat. De exponent wordt bekeken en het resultaat wordt in register A en de vlaggen gezet :

nul .... A = 00H, vlag Z, NC
positief ... A = 01H, vlag NZ, NC
negatief ... A = FFH, vlag NZ, C

adres 2E7DH

Deze routine zet enkel de exponent in DAC op nul.

adres 2E82H

Deze routine wordt gebruikt door de Factor Evaluator om de "ABS"-functie toe te passen op een operand in DAC. Eerst wordt het teken van de operand gecontroleerd (2EA1H). Is het positief, dan stopt de routine. Zoniet, wordt het type van de operand gecontroleerd via de standaardroutine GETYPR. Is de operand een string, dan wordt de foutmelding "Type mismatch" gegeven (406DH). Is het een geheel getal, dan wordt het teken ervan omgekeerd (322BH). Is het een getal met enkele of dubbele precisie, dan wordt het tekenbit van de mantisse in DAC omgekeerd.

adres 2E97H

Deze routine wordt door de Factor Evaluator gebruikt om de "SGN"-functie toe te passen op een operand in DAC. Het teken van de operand wordt bekeken (2EA1H), in registerpaar HL overgebracht en dan als een geheel getal in DAC gezet :

nul 0000H
positief ... 0001H
negatief ... FFFFH

adres 2EA1H

Deze routine berekent het teken van een operand in DAC. Eerst wordt het type gecontroleerd via de standaardroutine GETYPR. Indien het een string is, wordt de foutmelding "Type mismatch" gegeven (406DH). Als het een operand met enkele of dubbele precisie is, wordt het teken van de mantisse bekeken (2E71H). Is het een geheel getal, dan wordt de waarde ervan uit DAC+2 gehaald, en vertaald in een bepaalde vlaggen-stand, zoals bij de routine op 2E71H.

adres 2EB1H

Deze routine zet een operand met enkele precisie van vier bytes lang, die in DAC staat, op de stack van de Z80.

adres 2EC1H

Deze routine copieert de inhoud van registers C, B, E en D naar DAC.

adres 2ECCH

Deze routine copieert de inhoud van DAC in de registers C, B, E en D.

adres 2ED6H

Deze routine laadt de registers C, B, E en D met de inhoud van opeenvolgende stijgende adressen, te beginnen bij het adres dat in registerpaar HL staat.

adres 2EDFH

Deze routine laadt de registers E, D, C en B met de inhoud van opeenvolgende stijgende adressen, te beginnen bij het adres dat in registerpaar HL staat.

adres 2EE8H

Deze routine copieert een operand met enkele precisie van DAC naar het adres dat in registerpaar HL staat.

adres 2EEFH

Deze routine copieert elke operand van het adres dat in registerpaar HL staat, naar ARG. De lengte van de operand wordt uit VALTYP gehaald : 2 - geheel getal, 3 - string, 4 - enkele precisie, 8 - dubbele precisie.

adres 2F05H

Deze routine copieert elke operand van ARG naar DAC. De lengte van de operand wordt uit VALTYP gehaald : 2 = geheel getal, 3 = string, 4 = enkele precisie, 8 = dubbele precisie.

adres 2F0DH

Deze routine copieert elke operand van DAC naar ARG. De lengte van de operand wordt uit VALTYP gehaald : 2 = geheel getal, 3 = string, 4 = enkele precisie, 8 = dubbele precisie.

adres 2F21H

Deze routine wordt door de Expression Evaluator gebruikt om de relatie tussen twee operands met enkele precisie te berekenen. De eerste operand staat in registers C, B, E en D en de tweede in DAC. Het resultaat wordt in register A en de vlaggen gezet :

operand 1 = operand 2 : A = 00H, vlaggen Z, NC

operand 1 < operand 2 : A = 01H, vlaggen NZ, NC

operand 1 > operand 2 : A = FFH, vlaggen NZ, C

Opgemerkt dient te worden, dat voor relationele bewerkingen de Expression Evaluator maximaal negatieve getallen als klein, en maximaal positieve getallen als groot beschouwt.

adres 2F4DH

Deze routine wordt door de Expression Evaluator gebruikt om de relatie (><=) te berekenen tussen twee gehele getallen. Het eerste ervan staat in registerpaar DE, en het tweede in registerpaar HL. De resultaten worden weergegeven zoals bij de bewerkingen met getallen met enkele precisie (2F21H).

adres 2F83H

Deze routine wordt door de Expression Evaluator gebruikt om de relatie (><=) te berekenen tussen twee getallen met dubbele precisie. Het eerste ervan staat in DAC en het tweede in ARG. De resultaten worden weergegeven zoals bij de bewerkingen met getallen met enkele precisie (2F21H).

adres 2F8AH

Deze routine wordt door de Factor Evaluator gebruikt om de "CINT"-functie toe te passen op een operand in DAC. Eerst wordt het type van de operand gecontroleerd via de standaardroutine GETYPR. Is het al een geheel getal, dan stopt de routine. Is het een string, dan wordt de foutmelding "Type mismatch" gegeven (406DH). Is het een operand met enkele of dubbele precisie, dan wordt die omgezet in een binair geheel getal met teken, in registerpaar DE gezet (305DH) en dan als geheel getal in DAC gezet. Waarden die buiten het bereik van de machine vallen, leveren de foutmelding "Overflow" op (4067H).

adres 2FB2H

Deze routine wordt door de Factor Evaluator gebruikt om de "CSNG"-functie uit te voeren op een operand in DAC. Eerst wordt het type van de operand gecontroleerd via de standaardroutine GETYPR. Als hij al in enkele precisie staat, stopt de routine. Als het een string is, wordt de foutmelding "Type mismatch" gegeven (406DH). Als hij in dubbele precisie staat, wordt VALTYP gewijzigd (3053H) en de mantisse vanaf het zevende cijfer opgerond (2741H). Als de operand een geheel getal is, wordt het van binair naar een maximum van vijf BCD-cijfers omgezet door opeenvolgende delingen door de constanten 10000, 1000, 100, 10 en 1. Deze cijfers worden in DAC gezet, waar ze een mantisse met enkele precisie vormen. De exponent is gelijk aan het aantal significante cijfers in de mantisse. Indien er bijvoorbeeld vijf significante cijfers waren, dan werd de exponent 10^5.

adres 3030H TABEL

Deze tabel bevat de vijf constanten die door de vorige routine gebruikt worden : -10000, -1000, -100, -10, -1.

adres 303AH

Deze routine wordt door de Factor Evaluator gebruikt om de "CDBL"-functie uit te voeren op een operand in DAC. Eerst wordt het type van de operand gecontroleerd via de standaardroutine GETYPR. Is het al een getal met dubbele precisie, dan stopt de routine. Is het een string, dan wordt de foutmelding "Type mismatch" gegeven (406DH). Is het een geheel getal, dan wordt het eerst in enkele precisie omgezet (2FC8H), dan worden de laagste acht cijfers nul gemaakt, en VALTYP met 8 geladen.

adres 3058H

Deze routine controleert of een bepaalde operand een string is. Is dat niet zo, dan wordt de foutmelding "Type mismatch" gegeven (406DH).

adres 305DH

Deze routine wordt door de "CINT"-routine gebruikt om een BCD operand met enkele of dubbele precisie om te zetten in een binair geheel getal in het registerpaar DE. Vlag C wordt geset indien er een overflow optreedt.

Te beginnen met het grootste, worden alle cijfers van de mantisse één na één bij het product opgeteld. Na elke optelling wordt het product met 10 vermenigvuldigd. Hoeveel cijfers verwerkt moeten worden, bepaalt de exponent : is die bijvoorbeeld 10^5, dan worden 5 cijfers verwerkt. Tenslotte wordt het teken van de mantisse gecontroleerd en wordt, indien nodig, het product negatief gemaakt (3221H).

adres 30BEH

Deze routine wordt door de Factor Evaluator gebruikt om de "FIX"-functie toe te passen op een operand in DAC. Eerst wordt het type bepaald via de standaardroutine GETYPR. Is het een geheel getal, dan stopt de routine. Zoniet, wordt het teken van

de mantisse bekeken (2E71H) : is het positief, dan wordt de controle overgedragen aan de "INT"-routine (30CFH). Zoniet, wordt het getal positief gemaakt, de "INT"-functie uitgevoerd (30CFH) en het teken opnieuw omgekeerd.

adres 30CFH

Deze routine wordt door de Factor Evaluator gebruikt om de "INT"-functie uit te voeren op een operand in DAC. Eerst wordt het type bepaald via de standaardroutine GETYPR. Is het een geheel getal, dan stopt de routine. Het aantal cijfers na de komma wordt berekend door de exponent af te trekken van het aantal cijfers dat dit type operand telt, 6 voor enkele precisie, 14 voor dubbele precisie.

Indien de mantisse positief is, worden die cijfers na de komma gewoon op nul gezet. Was de mantisse negatief, dan wordt elk cijfer na de komma bekeken voordat het nul gemaakt wordt. Waren alle cijfers voordien nul, dan stopt de routine. Zoniet wordt -1,0 bij de operand opgeteld, via de optelroutine voor enkele precisie (324EH) of de routine voor dubbele precisie (269AH). Normaal wordt het type van een operand door de "CINT"-functie niet gewijzigd.

adres 314AH

Deze routine vermenigvuldigt de binaire gehele getallen zonder teken, die in registerparen BC en DE staan. Het resultaat komt in registerpaar DE. De gewone methode van schuiven en optellen, wordt gebruikt. Het product wordt zoveel maal met twee vermenigvuldigd, en de inhoud van registerpaar BC erbij opgeteld, als er bits "1" zijn in registerpaar DE. Deze routine wordt gebruikt door de Variabelen-zoekroutine op 5EA4H, om de positie van een element in een array te bepalen. Indien een overflow ontstaat, wordt de foutmelding "Subscript out of range" gegeven (601DH).

adres 3167H

Deze routine wordt door de Expression Evaluator gebruikt om twee gehele operands van mekaar af te trekken. De eerste wordt in registerpaar DE verwacht, de tweede in registerpaar HL. Het resultaat wordt in DAC gezet. De tweede operand wordt negatief gemaakt (3221H) en de controle gaat over naar de optel-routine.

adres 3172H

Deze routine wordt door de Expression Evaluator gebruikt om twee gehele operands bij elkaar op te tellen. De eerste wordt in registerpaar DE verwacht, de tweede in registerpaar HL. Het resultaat wordt in DAC gezet. Normaal worden beide binaire operands met teken gewoon opgeteld en in DAC gezet. Indien er evenwel een overflow optrad, worden ze beide omgezet in enkele precisie (2FCBH) en de controle wordt overgedragen aan de optelroutine voor enkele precisie-getallen op 324EH). Een overflow treedt op indien beide operands hetzelfde teken hebben, en het resultaat een ander teken heeft, bijvoorbeeld :

30000 + 15000 = -20536

adres 3193H

Deze routine wordt door de Expression Evaluator gebruikt om twee gehele operands te vermenigvuldigen. De eerste wordt in registerpaar DE verwacht, de tweede in registerpaar HL. Het resultaat wordt in DAC gezet. De beide tekens worden tijdelijk opgeslagen, en beide operands worden positief gemaakt (3215H). De vermenigvuldiging gebeurt op de gewone manier, met binair schuiven en optellen. Registerpaar HL wordt als product-accumulator gebruikt, terwijl registerpaar BC de eerste operand bevat en registerpaar DE de tweede. Wordt op een bepaald ogenblik tijdens de bewerking het product groter dan 7FFFH, dan worden beide operands omgezet in enkele precisie (2FCBH) en de controle overgedragen aan de routine die getallen met enkele precisie vermenigvuldigt (325CH). Blijft het product kleiner dan 7FFFH, dan worden de oorspronkelijke tekens teruggezet, en als die verschillend zijn, wordt het product negatief gemaakt. Het resultaat wordt, als een geheel getal, in DAC gezet (321DH).

adres 31E6H

Deze routine wordt door de Expression Evaluator gebruikt om het geheel quotient te berekenen van de deling van twee gehele operands (\). De eerste operand wordt in registerpaar DE verwacht, de tweede in registerpaar HL. Het resultaat wordt in DAC gezet. Is de tweede operand nul, dan wordt de foutmelding "Division by zero" gegeven (4058H). Zoniet, worden de tekens opgeslagen en beide operands positief gemaakt (3215H). De deling gebeurt op de gewone manier : binair schuiven en aftrekken, waarbij registerpaar HL de rest bevat, registerpaar BC de tweede operand en registerpaar DE de eerste operand en het product. Na de deling worden de tekens gezet zoals bij het begin en, als ze verschillen, wordt het product negatief gemaakt en vervolgens als een geheel getal in DAC gezet (321DH).

adres 3215H

Deze routine wordt gebruikt om twee binaire gehele getallen met een teken, in registerparen HL en DE, positief te maken. Beide oorspronkelijke tekens worden als een vlag in bit 7 van register B gezet : is dat bit 0, dan waren de tekens gelijk; is het bit 1, dan waren ze verschillend. Elke operand wordt nu bekeken en als hij negatief is wordt hij positief gemaakt door hem van nul af te trekken.

adres 322BH

Deze routine wordt door de "ABS"-functie gebruikt om een negatief geheel getal in DAC, positief te maken. Het negatief getal wordt uit DAC gehaald, het teken omgekeerd, en het wordt terug in DAC gezet (3221H). Als de waarde van de operand 8000H is, wordt hij omgezet naar enkele precisie (2FCCH) : de machine kent het gehele getal +32768 niet.

adres 323AH

Deze routine wordt door de Expression Evaluator gebruikt om de "MOD"-bewerking uit te voeren op twee gehele operands. De eerste ervan wordt in registerpaar DE verwacht, de tweede in registerpaar HL. Het resultaat wordt in DAC gezet. Het teken van

de eerste operand wordt opgeslagen en de twee operands worden gedeeld (31E6H). De rest wordt door de deling verdubbeld : daarom wordt registerpaar DE één bit naar rechts geschoven, om het te halveren. Het teken van de eerste operand wordt dan teruggehaald en als dat negatief was, wordt het teken van de rest omgekeerd. Dan wordt het in DAC gezet, als een geheel getal (321DH).

adres 324EH

Deze routine wordt door de Expression Evaluator gebruikt om twee operands met enkele precisie bij elkaar op te tellen. De eerste wordt in registers C, B, E en D verwacht, en de tweede in DAC. Het resultaat wordt in DAC gezet. De eerste operand wordt naar ARG gecopieerd (3280H), de tweede wordt omgezet naar dubbele precisie (3042H) en de controle gaat over op de optelroutine voor dubbele precisie (269AH).

adres 3257H

Deze routine wordt door de Expression Evaluator gebruikt om twee operands met enkele precisie van elkaar af te trekken. De eerste wordt in registers C, B, E en D verwacht, de tweede in DAC. Het resultaat wordt in DAC gezet. De tweede operand wordt negatief gemaakt (2E8DH) en de controle gaat over naar de routine om twee getallen met enkele precisie op te tellen (324EH).

adres 325CH

Deze routine wordt door de Expression Evaluator gebruikt om twee operands met enkele precisie te vermenigvuldigen. De eerste wordt verwacht in de registers C, B, E en D, de tweede in DAC. Het resultaat wordt in DAC gezet. De eerste operand wordt naar ARG gecopieerd (3280H). De tweede wordt in dubbele precisie omgezet, en de controle gaat over naar de routine die twee getallen met dubbele precisie vermenigvuldigt (27E6H).

adres 3265H

Deze routine wordt door de Expression Evaluator gebruikt om twee operands met enkele precisie te delen. De eerste wordt verwacht in de registers C, B, E en D, de tweede in DAC. Het resultaat wordt in DAC gezet. De beide operands worden van plaats gewisseld, zodat de eerste in DAC staat en de tweede in de genoemde registers. Dan wordt de tweede naar ARG gecopieerd (3280H). De eerste operand wordt in dubbele precisie omgezet (3042H) en de controle gaat over naar de routine die twee getallen met dubbele precisie deelt (289FH).

adres 3280H

Deze routine copieert de operand met enkele precisie die in de registers C, B, D en E staat, naar ARG en zet dan de vier minst significante bytes op nul.

adres 3299H

Deze routine zet een getal in tekstvorm (ASCII-karakters) om in één van de interne getal-types. Ze wordt gebruikt tijdens de omzetting in "tokens" en door de routines die "VAL", "INPUT" en "READ" verwerken. Bij de start van de routine bevat registerpaar HL het adres van het eerste karakter van de tekst-string die omgezet moet worden. Op het einde bevat registerpaar HL het adres van het karakter na die string. De numerieke operand staat in DAC en de code die het type van het getal bepaalt, in VALTYP. Een voorbeeld van de drie types :

XX	XX	FFH	7FH	XX	XX	XX	XX

Geheel getal : +32767

42H	17H	39H	04H	XX	XX	XX	XX

Enkele precisie : 0,173904*10^2

C2H	17H	39H	04H	62H	70H	93H	13H

Dubbele precisie : -0,17390462709313*10^2

Fig. 41 : types van getallen in DAC

Een geheel getal is een binair getal van zestien bits, in twee-complements-vorm. Het wordt opgeslagen in de volgorde : LSB eerst, MSB erna, te beginnen op DAC+2. Een geheel getal kan liggen tussen 8000H (-32768) en 7FFFH (+32767).

Een decimaal getal bestaat uit één byte voor de exponent en een mantisse van drie of zeven bytes. De exponent wordt opgeslagen in de vorm van een binair getal met een teken, en kan liggen tussen 01H (-63) over 40H (0) tot 7FH (+63). De speciale waarde 00H wordt gebruikt voor het getal 0. Deze exponentwaarden gelden voor een genormaliseerde mantisse. Getallen in exponentiële vorm worden door de Interpreter aan de gebruiker getoond met een cijfer ervoor. Daardoor is het bereik asymmetrisch : E-64 tot E+62. Bit 7 van de exponent bevat het teken van de mantisse : 0 = positief, 1 = negatief. De mantisse zelf wordt opgeslagen in "packed BCD"-vorm, met twee cijfers per byte. Merk op dat de Interpreter de inhoud van VALTYP gebruikt om het type van een getal te bepalen, niet het formaat van het getal.

De omzetting begint met het onderzoeken van het eerste karakter. Is dat een "&", dan wordt de controle overgedragen aan de routine die instaat voor het omrekenen van grondtallen (4EB8H). Is het een teken (+/-) dan wordt dat tijdelijk opgeslagen. Nu worden de opeenvolgende cijfers gelezen en opgeteld bij het gehele product, waarbij voor elk nieuw cijfer met tien wordt vermenigvuldigd. Wordt het product groter dan 32767, of wanneer een decimale punt gelezen wordt, wordt het product omgezet in enkele precisie en alle volgende karakters worden rechtstreeks in DAC gezet. Wordt een zevende cijfer gevonden, dan wordt het product omgezet in dubbele precisie. Cijfers na het veertiende cijfer worden nog wel gelezen, maar er wordt geen rekening mee

gehouden.
De omzetting wordt beëindigd wanneer er een niet-numeriek karakter gelezen wordt. Is dat laatste karakter een type-bepaling ("%", "#", of "!") dan wordt de overeenkomstige omzettingsroutine aangeroepen en de controle gaat over naar het einde van de routine (331EH). Was het een voorvoegsel voor een exponent ("E", "e", "D" of "d") dan wordt eveneens één van de omzettingsroutines aangeroepen, en daarna worden de volgende cijfers omgerekend naar een binaire exponent in register E.

Bij het eindpunt van de routine (331EH) wordt het type van het product gecontroleerd via de standaardroutine GETYPR. Is dat product een getal met enkele of dubbele precisie, dan wordt de exponent berekend : eerst wordt het aantal decimalen (in register B) afgetrokken van het totale aantal cijfers (in register D). Dat levert het aantal cijfers voor de komma op. Dit wordt opgeteld bij een expliciet opgegeven exponent, in register E, en in DAC+Ø gezet, als de exponent van het product.

Het karakter dat het teken aangaf wordt nu teruggehaald (2E86H). Als dat nodig is, wordt het product negatief gemaakt (2E86H). Is het product een geheel getal, dan stopt de routine. Is het een getal met enkele precisie, dan wordt nog eens gecontroleerd of het niet de waarde -32768 heeft (2FA2H). Is het product een getal met dubbele precisie, dan wordt dit vanaf het vijftiende cijfer opgerond (273CH), en de routine stopt.

adres 340AH

Deze routine wordt door de fout-verwerkingsroutine gebruikt om de tekst " in " op het scherm te zetten (6678H), gevolgd door een regelnummer dat in registerpaar HL wordt verwacht (3412H).

adres 3412H

Deze routine zet een binair geheel getal zonder teken, dat in registerpaar HL wordt aangebracht, op het scherm. De operand wordt als een geheel getal in DAC gezet (2F99H), omgezet naar tekst (ASCII-karakters) en op het scherm gezet (6677H).

adres 3425H

Deze routine zet de numerieke operand in DAC om in tekstvorm (ASCII-karakters) en zet die in FBUFFR. Het adres van het eerste karakter van dat getal in tekstvorm, staat op het einde van de routine in registerpaar HL. Op het einde van de reeks cijfers wordt een byte met inhoud Ø gezet.

Eerst wordt de operand in dubbele precisie omgezet (375FH). De BCD-cijfers van de mantisse worden daar dan uitgehaald, omgezet in ASCII-karakters, en in FBUFFR geplaatst (36B3H). De exponent bepaalt waar de decimale komma moet staan. Een voorbeeld :

```
0,999*10^+2 - 99,9
0,999*10^+1 -  9,99
0,999*10^+0 -  0,999
0,999*10^-1 -  0,0999
```

Is de exponent kleiner van 10^-1 of groter dan 10^+14, dan wordt het nummer in exponentiële vorm voorgesteld. De decimale komma

wordt na het eerste cijfer gezet, en de exponent wordt uit binaire notatie omgezet en volgt op de mantisse.

Een tweede toegangsadres is 3426H : dit wordt gebruikt wanneer het "PRINT USING"-statement wordt verwerkt. Wanneer dit startpunt wordt gebruikt, staat het aantal karakters voor de decimale komma in register B. Het aantal karakters erna, staat in register C, en in register A staat een coderings-byte :

7	6	5	4	3	2	1	0
1	,	*	£	+	teken	0	^

Fig. 42 : coderings-byte

Hier gebeurt er niet veel anders dan normaal, maar er zijn meer mogelijkheden. Nadat de operand in dubbele precisie is omgezet, zal het getal in exponentiële vorm worden geschreven indien bit 0 van register A geset is. De mantisse wordt naar rechts geschoven in DAC en opgerond, om ongewenste cijfers na de komma kwijt te raken (377BH). Bij het omzetten van de mantisse in ASCII-karakters (36B3H) worden, op de gepaste plaatsen, komma's tussen de cijfers gezet (het Angelsaksische equivalent voor de punten die in Europa gebruikt worden tussen elke groep van drie cijfers; de vertaler) indien bit 6 van register A geset is. Na de omzetting wordt het getal in de aangegeven vorm gegoten (351CH), waarbij ongebruikte plaatsen voor de komma opgevuld worden met asterisken indien bit 5 van het coderings-byte geset is. Staat bit 4 op "1", dan wordt voor het getal een Pond-symbool gezet (£). Staat bit 3 op "1" dan wordt een "+" voor positieve getallen gedrukt, anders een spatie. Bit 2 bepaalt de plaats van het teken van het getal : ervoor, indien het "0" is, erna indien het bit "1" is.

Een derde toegangsadres tot deze routine is 3441H. Dit stuk wordt gebruikt om gehele getallen zonder teken om te zetten in tekst-vorm, met name regelnummers. Het getal 9000H bijvoorbeeld, zou weergegeven worden als -28672 indien het als een gewoon getal werd omgezet. Door de routine op dit derde toegangspunt aan te roepen, wordt het omgezet in 36864. De operand wordt omgezet door opeenvolgende delingen door 10000, 1000, 100, 10 en 1, en de zo verkregen cijfers worden in FBUFFR geplaatst (36DBH).

adres 3710H TABEL

Deze tabel bevat de vijf constanten die gebruikt worden door de routine die cijfers op het scherm print : 10000, 1000, 1000, 10 en 1.

adres 371AH

Deze routine wordt gebruikt door de "BIN$"-functie, om een numerieke operand in DAC in tekstvorm om te zetten. Register B wordt geladen met de grootte van de reeks (1) en de controle gaat over naar de algemene omzettings-routine (3724H).

adres 371EH

Deze routine wordt gebruikt door de "OCT$"-functie, om een numerieke operand in DAC in tekstvorm om te zetten. Register B wordt geladen met de grootte van de reeks (3) en de controle wordt overgedragen aan de algemene omzettings-routine (3724H).

adres 3722H

Deze routine wordt gebruikt door de "HEX$"-functie om een numerieke operand in DAC in tekstvorm om te zetten. Register B wordt geladen met de grootte van de reeks (4). De operand wordt omgezet in een binair geheel getal, en in registerpaar HL gezet (5439H). Opeenvolgende groepen van 1, 3 of 4 bits worden naar rechts geschoven, in ASCII-cijfers omgezet en in FBUFFR gezet. Wanneer, door het herhaaldelijk schuiven, de operand nog enkel nullen bevat, stopt de routine. Het adres van het eerste cijfer staat in registerpaar HL, en het byte na de string wordt met nul geladen.

adres 3752H

Deze routine wordt gebruikt tijdens de output van numerieke gegevens, om in register B het aantal cijfers te zetten dat een operand telt, en het adres van het minst significante byte ervan in registerpaar HL. Voor getallen met enkele precisie bevat register B, 6 en registerpaar HL = DAC+3. Voor dubbele precisie bevat register B,14 en registerpaar HL : DAC+7.

adres 375FH

Deze routine wordt gebruikt tijdens de output van numerieke gegevens, om een numerieke operand in DAC in dubbele precisie om te zetten (303AH).

adres 377BH

Deze routine wordt gebruikt tijdens de output van numerieke gegevens, om de mantisse in DAC naar rechts te schuiven (27DBH). Daartoe wordt het omgekeerde van het aantal cijfers in register A gezet. Het resultaat wordt opgerond vanaf het vijftiende cijfer (2741H).

adres 37A2H

Deze routine wordt gebruikt tijdens de output van numerieke gegevens om het omgekeerde te berekenen van het aantal cijfers na de komma in een operand met vlottende komma. Dit gebeurt door de exponent af te trekken van het aantal cijfers dat de operand telt (6 of 14).

adres 37B4H

Deze routine wordt gebruikt tijdens de output van numerieke gegevens om de plaats te bepalen van het laatste van nul verschillende cijfer dat in de mantisse in DAC staat. Op het einde van de routine bevat registerpaar HL het adres van dat cijfer.

adres 37C8H

Deze routine wordt gebruikt door de Expression Evaluator voor een machtsverheffing van twee operands met enkele precisie. De eerste operand wordt verwacht in de registers C, B, E en D, en de tweede in DAC. Het resultaat wordt in DAC gezet. De eerste operand wordt naar ARG gecopieerd (3280H), op de stack gePUSHt (2CC7H) en verwisseld met DAC (2C6FH). Daarna wordt de tweede operand in ARG gePOPt, en de verdere verwerking gebeurt door de routine hierna.

adres 37D7H

Deze routine wordt door de Expression Evaluator gebruikt voor een machtsverheffing van twee operands met dubbele precisie. De eerste operand wordt in DAC verwacht, de tweede in ARG. Het resultaat wordt in DAC gezet. Gewoonlijk wordt het resultaat berekend met de formule :

X^P = EXP(P*LOG(X))

Er is een andere, veel snellere, oplossing indien de operand die de macht aanduidt, een geheel getal is. Dit wordt gecontroleerd door het gehele gedeelte van de operand te bepalen en dit te vergelijken met de oorspronkelijke waarde (39A1H). Zijn beide gelijk, dan wil dat zeggen dat de snellere manier gebruikt kan worden. Die wordt hieronder beschreven.

adres 383FH

Deze routine wordt door de Expression Evaluator gebruikt voor een machtsverheffing van twee gehele operands. De eerste wordt in registerpaar DE verwacht, de tweede in registerpaar HL. Het resultaat wordt in DAC gezet. De routine ontleedt het probleem in een reeks van eenvoudige vermenigvuldigingen :

6^13 = 6^1101 = (6^8)*(6^4)*(6^1)

Omdat de operand die de macht aanduidt, in binaire vorm staat, is een verschuiving naar rechts voldoende om te weten of een bepaald tussenproduct bij het resultaat opgeteld moet worden. Die tussenproducten zelf worden berekend door cumulatieve vermenigvuldiging van de operand, elke keer als de berekeningslus wordt doorlopen. Indien op een bepaald ogenblik het product te groot wordt (om als geheel getal opgeslagen te worden), wordt het in enkele precisie omgezet. Is de routine klaar, dan wordt de machtsfactor bekeken. Is die negatief, dan wordt de reciproke van het product berekend, omdat X^(-P) = 1/(X^P).

adres 390DH

Deze routine wordt bij het machtsverheffen gebruikt, om twee gehele getallen te vermenigvuldigen (3193H). Indien het resultaat een "overflow" geeft, en het dus in enkele precisie moet worden omgezet, wordt vlag Z gereset.

adres 391AH

Deze routine wordt bij het machtsverheffen gebruikt om te controleren of een operand in dubbele precisie, een geheel getal is. Is dit zo, dan wordt vlag C gereset.

adres 392EH TABEL

Deze tabel met adressen wordt door de Verwerkingslus van de Interpreter gebruikt om te bepalen waar een bepaald token (een bepaald statement dus) verwerkt wordt. Hoewel ze niet in de tabel staan, vermelden we hieronder ook de bijbehorende "keywords" :

ADRES	STATEMENT	ADRES	STATEMENT	ADRES	STATEMENT
63EAH	END	00C3H	CLS	5B11H	CIRCLE
4524H	FOR	51C9H	WIDTH	7980H	COLOR
6527H	NEXT	485DH	ELSE	5D6EH	DRAW
485BH	DATA	6438H	TRON	59C5H	PAINT
4B6CH	INPUT	6439H	TROFF	00C0H	BEEP
5E9FH	DIM	643EH	SWAP	73E5H	PLAY
4B9FH	READ	6477H	ERASE	57EAH	PSET
4880H	LET	49AAH	ERROR	57E5H	PRESET
47E8H	GOTO	495DH	RESUME	73CAH	SOUND
479EH	RUN	53E2H	DELETE	79CCH	SCREEN
49E5H	IF	49B5H	AUTO	7BE2H	VPOKE
63C9H	RESTORE	5468H	RENUM	7A48H	SPRITE
47B2H	GOSUB	4718H	DEFSTR	7B37H	VDP
4821H	RETURN	471BH	DEFINT	7BA5H	BASE
485DH	REM	471EH	DEFSNG	55A8H	CALL
63E3H	STOP	4721H	DEFDBL	7911H	TIME
4A24H	PRINT	4B0EH	LINE	786CH	KEY
64AFH	CLEAR	6AB7H	OPEN	7E4BH	MAX
522EH	LIST	7C52H	FIELD	73B7H	MOTOR
6286H	NEW	775BH	GET	6EC6H	BLOAD
48E4H	ON	7758H	PUT	6E92H	BSAVE
401CH	WAIT	6C14H	CLOSE	7C16H	DSKO$
501DH	DEF	6B5DH	LOAD	7C1BH	SET
5423H	POKE	6B5EH	MERGE	7C20H	NAME
6424H	CONT	6C2FH	FILES	7C25H	KILL
6FB7H	CSAVE	7C48H	LSET	7CA2H	IPL
703FH	CLOAD	7C4DH	RSET	7C2FH	COPY
4016H	OUT	6BA3H	SAVE	7C34H	CMD
4A1DH	LPRINT	6CA2H	LFILES	7766H	LOCATE
5229H	LLIST				

adres 39DEH TABEL

Deze tabel met adressen wordt door de Factor Evaluator gebruikt om de routine te localiseren waarin een bepaalde functie wordt verwerkt. Hoewel die niet in de tabel staan, geven we hieronder ook de bijbehorende "keywords" weer :

ADRES	FUNCTIE	ADRES	FUNCTIE	ADRES	FUNCTIE
6861H	LEFT$	4FCCH	POS	30BEH	FIX
6891H	RIGHT$	67FFH	LEN	7940H	STICK
689AH	MID$	6604H	STR$	794CH	STRIG
2E97H	SGN	68BBH	VAL	795AH	PDL
30CFH	INT	680BH	ASC	7969H	PAD
2E82H	ABS	681BH	CHR$	7C39H	DSKF
2AFFH	SQR	541CH	PEEK	6D39H	FPOS
2BDFH	RND	7BF5H	VPEEK	7C66H	CVI
29ACH	SIN	6848H	SPACE$	7C6BH	CVS
2A72H	LOG	65F5H	OCT$	7C70H	CVD
2B4AH	EXP	65FAH	HEX$	6D25H	EOF
2993H	COS	4FC7H	LPOS	6D03H	LOC
29FBH	TAN	65FFH	BIN$	6D14H	LOF
2A14H	ATN	2F8AH	CINT	7C57H	MKI$
69F2H	FRE	2FB2H	CSNG	7C5CH	MKS$
4001H	INP	303AH	CDBL	7C61H	MKD$

adres 3A3EH TABEL

Deze adressentabel wordt tijdens de omzetting in tokens gebruikt als een soort wegwijzer in de tabel met BASIC keywords (3A72H). Elk van de zesentwintig adressen in de tabel is het startadres van een blok in de tabel met keywords. Het eerste adres verwijst naar de keywords die met "A" beginnen, het tweede naar de groep die met "B" begint, en zo verder.

ADRES	TOKEN	ADRES	TOKEN	ADRES	TOKEN
3A72H	A	3B9FH	J	3C8EH	S
3A88H	B	3BA0H	K	3CDBH	T
3A9FH	C	3BABH	L	3CF6H	U
3AF3H	D	3BE8H	M	3CFFH	V
3B2EH	E	3C09H	N	3D16H	W
3B4FH	F	3C18H	O	3D20H	X
3B69H	G	3C2BH	P	3D24H	Y
3B7BH	H	3C5DH	Q	3D25H	Z
3B80H	I	3C5EH	R		

adres 3A72H TABEL

Deze tabel bevat de keywords in BASIC, en de overeenkomstige tokens. Elk blok in de tabel bevat alle keywords die met een bepaalde letter beginnen; een blok wordt afgesloten met een byte die 0 bevat. De keywords staan als tekst-strings in de tabel, met bit 7 van het laatste karakter geset, om het einde aan te geven, direkt gevolgd door het overeenkomstige token. Het was niet nodig om het eerste karakter in de tabel te zetten, aangezien dit door de positie in de tabel wordt bepaald. Hieronder volgt de volledige lijst. De blokken voor "J", "Q", "Y" en "Z" zijn leeg.
(zie tabel op de volgende bladzijde)

ADRES	TOKEN	ADRES	TOKEN	ADRES	TOKEN	ADRES	TOKEN
AUTO	A9H	DSKF	26H	LIST	93H	REM	8FH
AND	F6H	DRAW	BEH	LFILES	BBH	RESUME	A7H
ABS	06H	ELSE	A1H	LOG	0AH	RSET	B9H
ATN	0EH	END	81H	LOC	2CH	RIGHT$	02H
ASC	15H	ERASE	A5H	LEN	12H	RND	08H
ATTR$	E9H	ERROR	A6H	LEFT$	01H	RENUM	AAH
BASE	C9H	ERL	E1H	LOF	2DH	SCREEN	C5H
BSAVE	D0H	ERR	E2H	MOTOR	CEH	SPRITE	C7H
BLOAD	CFH	EXP	0BH	MERGE	B6H	STOP	90H
BEEP	C0H	EOF	2BH	MOD	FBH	SWAP	A4H
BIN$	1DH	EQV	F9H	MKI$	2EH	SET	D2H
CALL	CAH	FOR	82H	MKS$	2FH	SAVE	BAH
CLOSE	B4H	FIELD	B1H	MKD$	30H	SPC(	DFH
COPY	D6H	FILES	B7H	MID$	03H	STEP	DCH
CONT	99H	FN	DEH	MAX	CDH	SGN	04H
CLEAR	92H	FRE	0FH	NEXT	83H	SQR	07H
CLOAD	9BH	FIX	21H	NAME	D3H	SIN	09H
CSAVE	9AH	FPOS	27H	NEW	94H	STR$	13H
CSRLIN	E8H	GOTO	89H	NOT	E0H	STRING$	E3H
CINT	1EH	GO TO	89H	OPEN	B0H	SPACE$	19H
CSNG	1FH	GOSUB	8DH	OUT	9CH	SOUND	C4H
CDBL	20H	GET	B2H	ON	95H	STICK	22H
CVI	28H	HEX$	1BH	OR	F7H	STRIG	23H
CVS	29H	INPUT	85H	OCT$	1AH	THEN	DAH
CVD	2AH	IF	8BH	OFF	EBH	TRON	A2H
COS	0CH	INSTR	E5H	PRINT	91H	TROFF	A3H
CHR$	16H	INT	05H	PUT	B3H	TAB(	DBH
CIRCLE	BCH	INP	10H	POKE	98H	TO	D9H
COLOR	BDH	IMP	FAH	POS	11H	TIME	CBH
CLS	9FH	INKEY$	ECH	PEEK	17H	TAN	0DH
CMD	D7H	IPL	D5H	PSET	C2H	USING	E4H
DELETE	A8H	KILL	D4H	PRESET	C3H	USR	DDH
DATA	84H	KEY	CCH	POINT	EDH	VAL	14H
DIM	86H	LPRINT	9DH	PAINT	BFH	VARPTR	E7H
DEFSTR	ABH	LLIST	9EH	PDL	24H	VDP	C8H
DEFINT	ACH	LPOS	1CH	PAD	25H	VPOKE	C6H
DEFSNG	ADH	LET	88H	PLAY	C1H	VPEEK	18H
DEFDBL	AEH	LOCATE	D8H	RETURN	8EH	WIDTH	A0H
DSKO$	D1H	LINE	AFH	READ	87H	WAIT	96H
DEF	97H	LOAD	B5H	RUN	8AH	XOR	F8H
DSKI$	EAH	LSET	B8H	RESTORE	8CH		

adres 3D26H TABEL

Deze tabel van 21 bytes lang wordt door de Interpreter gebruikt tijdens de omzetting in tokens. Hij bevat de tien keywords die maar uit één karakter bestaan, samen met de overeenkomstige tokens.

```
+ ... F1H   * ... F3H   ^ ... F5H   ' ... E6H   = ... EFH
- ... F2H   / ... F4H   \ ... FCH   > ... EEH   > ... F0H
```

adres 3D3BH TABEL

Deze tabel wordt door de Expression Evaluator gebruikt om het voorrangs-niveau te bepalen van een bepaalde bewerking tussen twee operands. Hoe groter de waarde in de tabel, hoe hoger de voorrang van een bepaalde bewerking. In de tabel staan niet de waarden voor de relationele bewerkingen (64H), voor de "NOT"-bewerking (5AH) en de negatie (7DH). Die drie worden rechtstreeks door de Expression Evaluator en de Factor Evaluator bepaald.

```
79H ... +          46H ... OR
79H ... -          3CH ... XOR
7CH ... *          32H ... EQV
7CH ... /          28H ... IMP
7FH ... ^          7AH ... MOD
50H ... AND        7BH ... \
```

adres 3D47H TABEL

Deze tabel wordt gebruikt om het resultaat van een functie die door de gebruiker zelf werd bepaald, om te zetten in hetzelfde type van variabele als in die definitie werd gebruikt. De tabel bevat de adressen van de routines die types van variabelen omzetten :

```
303AH ... CDBL
0000H ... niet gebruikt
2F8AH ... CINT
3058H ... controleer of het een string is
2FB2H ... CSNG
```

adres 3D51H TABEL

Deze adressentabel wordt door de Expression Evaluator gebruikt om op te zoeken door welke routine een bepaalde wiskundige bewerking wordt uitgevoerd, wanneer beide operands in dubbele precisie staan.

```
269AH ... +
268CH ... -
27E6H ... *
289FH ... /
37D7H ... ^
2F83H ... relatie
```

adres 3D5DH TABEL

Deze adressentabel wordt door de Expression Evaluator gebruikt om op te zoeken door welke routine een bepaalde wiskundige bewerking wordt uitgevoerd, wanneer beide operands in enkele precisie staan.

```
324EH ... +
3257H ... -
325CH ... *
3267H ... /
37C8H ... ^
2F21H ... relatie
```

adres 3D69H TABEL

Deze adressentabel wordt door de Expression Evaluator gebruikt om op te zoeken door welke routine een bepaalde wiskundige bewerking wordt uitgevoerd, wanneer beide operands gehele getallen zijn.

- 3172H ... +
- 3167H ... -
- 3193H ... *
- 4DB8H ... /
- 383FH ... ^
- 2F4DH ... relatie

adres 3D75H TABEL

Deze tabel bevat de foutmeldingen die de Interpreter geeft. Ze staan allemaal in de tabel als tekst-strings, met een 0 erachter om het einde aan te geven. De bijbehorende foutcodes worden hieronder enkel uit oogpunt van volledigheid gegeven : die staan niet in de tabel.

01	NEXT without FOR	19	Device I/O error
02	Syntax error	20	Verify error
03	RETURN without GOSUB	21	No RESUME
04	Out of DATA	22	RESUME without error
05	Illegal function call	23	Unprintable error
06	Overflow	24	Missing operand
07	Out of memory	25	Line buffer overflow
08	Undefined line number	50	FIELD overflow
09	Subscript out of range	51	Internal error
10	Redimensioned array	52	Bad file name
11	Division by zero	53	File not found
12	Illegal direct	54	File already open
13	Type mismatch	55	Input past end
14	Out of string space	56	Bad file name
15	String too long	57	Direct statement in file
16	String formula too complex	58	Sequential I/O only
17	Can't CONTINUE	59	File not OPEN
18	Undefined user function		

adres 3FD2H

Hier staat gewoon de tekst " in ", gevolgd door een 0.

adres 3FD7H

Hier staat gewoon de tekst "Ok", een "CR"-code, een "LF"-code en een nul.

adres 3FDCH

Hier staat gewoon de tekst "Break", gevolgd door een 0.

adres 3FE2H

Deze routine doorzoekt de stack van de Z80 naar een blok dat parameters van een "FOR"-lus bevat. Het adres van de

lusvariabele wordt in registerpaar DE verwacht. Het zoeken begint vier bytes hoger dan het adres in de Stack Pointer van de Z80 : daaronder staan de "RET"-adressen van de aanroepende routine en van de Verwerkingslus. Wordt geen "FOR"-code (82H) gevonden, dan stopt de routine met vlag NZ. Wordt een "FOR"-code gevonden, dan wordt het adres van de lusvariabele uit het blok gehaald en vergeleken met dat in registerpaar DE. Zijn die gelijk, dan stopt de routine met vlag Z en met in registerpaar HL het adres van het byte in het parameterblok dat het type van variabele aangeeft. Waren de lusvariabelen verschillend, dan gaat het zoeken verder, tweeëntwintig bytes verderop in de stack, de start van het volgende parameterblok.

adres 4001H

Deze routine wordt door de Factor Evaluator gebruikt om de "INP"-functie toe te passen op een operand in DAC. Het nummer van de poort wordt gecontroleerd (5439H), de poort wordt uitgelezen en het resultaat wordt, als een geheel getal, in DAC gezet (4FCFH).

adres 400BH

Deze routine evalueert een operand tussen -32768 en +65535 (542FH) en zet die in registerpaar BC. Daarna wordt, via de standaardroutine SYNCHR, gecontroleerd of een tweede operand, tussen 0 en 255, een komma bevat; die wordt geëvalueerd (521CH) en in register A gezet.

adres 4016H

Deze routine verwerkt het "OUT"-statement. Het poortnummer en het databyte worden geëvalueerd (400BH) en dan wordt het databyte naar de vermelde poort van de Z80 geschreven.

adres 401CH

Deze routine verwerkt het "WAIT"-statement. Eerst worden het poortnummer en de operands die bij "AND" horen, geëvalueerd (400BH), en daarna de eventuele operands bij "XOR" (521CH). Dan wordt de poort herhaaldelijk uitgelezen, en de uitlezing vergeleken met de operands (eerst "XOR", dan "AND"), tot het resultaat verschilt van nul. In tegenstelling tot wat in bepaalde MSX-handboeken wordt geschreven, kan deze lus wél onderbroken worden door "CTRL/STOP" in te drukken, omdat in de lus de standaardroutine CKCNTC wordt aangeroepen.

adres 4039H

Deze routine wordt door de Verwerkingslus gebruikt, wanneer die op het einde van de programma-tekst gekomen is. ONEFLAG wordt gecontroleerd, om te zien of er een fout-code in staat. Is dat zo, dan wordt de foutmelding "No RESUME" gegeven. Zoniet, wordt het programma op de gewone manier beëindigd (6401H). De bedoeling van deze routine is om een "ON ERROR"-routine zonder "RESUME" op het einde, op te sporen.

adres 404FH

Deze routine wordt gebruikt door de routine die het

"READ"-statement verwerkt, wanneer in een "DATA"-statement een fout wordt geconstateerd. Het regelnummer dat in DATLIN staat, wordt gecopieerd in CURLIN. Daardoor zal de foutmeldings-routine de "DATA"-regel in plaats van een bepaalde programmaregel opgeven als de regel die een fout bevat. Daarna gaat de controle over naar de "Syntax Error"-verwerker

adres 4055H TABEL

Op dit adres staan negen foutcodes. Register E wordt geladen met één ervan, en de controle loopt door in de foutverwerker.

ADRES	FOUTMELDING
4055H	Syntax error
4058H	Division by zero
405BH	NEXT without FOR
405EH	Redimensioned array
4061H	Undefined user function
4064H	RESUME without error
4067H	Overflow error
406AH	Missing operand
406DH	Type mismatch

adres 406FH

Deze routine verwerkt de fouten die de Interpreter ontdekt. Alle foutcodes komen hier terecht, met de code in register E. Eerst wordt in VLZADR gekeken of de tekst van het programma gewijzigd werd door de routine die het "VAL"-statement verwerkt. Was dat zo, dan wordt het oorspronkelijke karakter uit VLZDAT gehaald en teruggezet. Nu wordt het regelnummer dat momenteel werd verwerkt, uit CURLIN naar ERRLIN en DOT gecopieerd. De stack van de Z80 wordt hersteld met behulp van SAVSTK (62F0H). De foutcode wordt in ERRFLG gezet, waar ze door een "ERR"-functie opgehaald kan worden. De plaats in het programma, waar de Interpreter op dat ogenblik was, wordt uit SAVTXT naar ERRTXT gecopieerd, ten behoeve van de routine die het "RESUME"-statement verwerkt. Het nummer van de regel en de plaats in de tekst waar de fout optrad, worden eveneens naar OLDLIN en OLDTXT gecopieerd, ten behoeve van de routine die het "CONT"-statement verwerkt. Via ONELIN wordt gecontroleerd of voordien een "ON ERROR"-statement werd verwerkt. Was dat zo, en op voorwaarde dat er niet reeds een foutcode actief was, dan wordt de controle overgedragen aan de Verwerkingslus (4620H) die dan de BASIC regels zal uitvoeren waar het "ON ERROR"- statement naar verwijst.

Was er geen "ON ERROR"-statement actief, dan wordt de foutcode gebruikt om in de tabel met foutmeldingen op 3D75H de gepaste tekst op te zoeken. Er wordt een "CR"-code en een "LF"-code doorgegeven (7323H) en het scherm wordt terug naar tekstmode gebracht via TOTEXT. Dan wordt een "BELL"-code doorgegeven en de foutmelding gegeven (6678H). Indien de Interpreter een programma aan het verwerken was (dus niet een direkt commando), wordt die foutmelding gevolgd door het regelnummer (340AH), en de controle loopt door in de "Ok"-routine.

adres 411FH

Dit is het toegangsadres tot de Hoofdlus, bij de beëindiging van een programma. Het scherm wordt in tekstmode gezet via de standaardroutine IOTEXT, de printer wordt vrijgemaakt (7304H) en In/Out-buffer Ø wordt gesloten (6D7BH). De codes voor "CR" en "LF" worden naar het scherm gestuurd (7323H), de tekst "Ok" wordt afgedrukt (6678H) en de controle loopt door in de Hoofdlus.

adres 4134H

Dit is de hoofdlus van de Interpreter. Eerst wordt CURLIN op FFFFH gezet, om aan te geven dat de Interpreter in directe mode werkt (d.w.z. er wordt geen programma verwerkt, maar gegevens die rechtstreeks worden ingetypt). Dan wordt in AUTFLG gecontroleerd of de "AUTO"-mode actief is. Is dat zo, dan wordt uit AUTLIN het volgende regelnummer gehaald en op het scherm gezet (3412H). Het programmageheugen wordt doorzocht, om te zien of die regel al voorkwam (4295H) en op het scherm komt een asterisk of een spatie, al naargelang.

De standaardroutine ISFLIO bekijkt dan of een "LOAD"-statement verwerkt wordt. Zo ja, wordt de programmaregel van de cassette gelezen (7374H), zo neen wordt hij van het toetsenbord gehaald, via de standaardroutine PINLIN. Is de regel leeg, of werd "CTRL/STOP" ingedrukt, dan gaat de controle terug naar de start van de Hoofdlus (4134H), zonder meer. Begint de regel met een regelnummer, dan wordt dit omgezet in een geheel getal zonder teken, en in registerpaar DE gezet (4769H). De hele regel wordt vervolgens gecodeerd en van "tokens" voorzien, en in KBUF gezet (42B2H). Was er geen regelnummer aanwezig, dan wordt de controle overgedragen aan de Verwerkingslus (6D48H), die de regel zal uitvoeren.

In de veronderstelling dat een regel met een regelnummer begint, wordt er gekeken of hij leeg is. Het resultaat van die test wordt tijdelijk opgeslagen. Het regelnummer wordt naar DOT gecopieerd en AUTLIN wordt verhoogd met wat in AUTINC staat. Wordt AUTLIN daardoor hoger dan 65530, dan wordt de "AUTO"-mode afgezet. Het programmageheugen wordt nu doorlopen (4295H) op zoek naar dat regelnummer of, als het niet gevonden wordt, naar de plaats van het eerstvolgende regelnummer. Wordt het regelnummer niet gevonden én de regel is leeg én de "AUTO"-mode is niet actief, dan wordt de foutmelding "Undefined line number" gegeven (481CH). Wordt het regelnummer wel gevonden én de regel is leeg én de "AUTO"-mode is actief, dan gaat de Hoofdlus gewoon verder met het volgende statement (4237H).

In het andere geval worden alle pointers in het programmageheugen terug in regelnummers omgezet (54EAH), en een eventueel bestaande programmaregel wordt gewist (5405H). Als de nieuwe regel niet leeg is, wordt in het programmageheugen voldoende ruimte vrijgemaakt (6250H) en de gecodeerde programmaregel wordt daarnaar gecopieerd vanuit KBUF.

De schakeladressen in het programmageheugen worden opnieuw berekend (4257H), het variabelengeheugen leeggemaakt (629AH) en de controle gaat terug naar de start van de Hoofdlus.

adres 4253H

Deze routine herberekent de schakeladressen in het programmageheugen, na een wijziging in het programma. De eerste twee bytes van elke programmaregel bevatten het startadres van de volgende regel : dit wordt het schakeladres genoemd. Het gebruik ervan vergt weliswaar meer geheugenruimte per regel, maar maakt dat de Interpreter de volgende regel veel sneller kan vinden.

Een voorbeeld van een programmaregel staat hieronder. De regel "10 PRINT 9" staat in dit geval in het begin van het programmageheugen (8001H) :

09H	80H	0AH	00H	91H	20H	1AH	00H

Fig. 43 : programmaregel

In het voorbeeld staat het schakeladres 8009H, in de volgorde zoals dat bij de Z80 gebruikelijk is : LSB, MSB. Daarna komt het regelnummer, in diezelfde volgorde geschreven. Het statement zelf bestaat uit een "PRINT"-token (91H), een spatie (20H), het getal negen (1AH) en het karakter 00H, dat het einde van de regel aangeeft. Voor meer details over de manier waarop een programmaregel wordt opgeslagen, verwijzen we naar de routine die een regel codeert en van tokens voorziet (42B2H).

Elk schakeladres wordt opnieuw berekend door gewoon de hele regel af te zoeken tot het eind-karakter 00H wordt gevonden. De routine stopt op het ogenblik dat het einde van het programmageheugen wordt bereikt. Dit punt is te herkennen aan het speciale schakeladres 0000H.

adres 4279H

Deze routine wordt gebruikt door de routine die het "LIST"-statement verwerkt. Ze leest uit de programmatekst tot maximaal twee operands uit, die regelnummers moeten zijn. Is een eerste regelnummer aanwezig, dan wordt het omgezet in een geheel getal zonder teken, in registerpaar DE (475FH). Zoniet, wordt de waarde 0000H genomen. Het eventuele tweede regelnummer moet door een "-"-token (F2H) worden voorafgegaan. Dit nummer wordt op de stack van de Z80 gezet. Is er geen tweede nummer, dan wordt de waarde 65530 aangenomen. Daarna loopt de routine door in de routine die het programmageheugen doorzoekt, om de eerst vermelde regel te vinden.

adres 4295H

Deze routine doorzoekt het programmageheugen naar de programmaregel waarvan het regelnummer in registerpaar DE staat. Te beginnen bij het adres in TXTTAB, wordt elke programmaregel doorlopen en het regelnummer vergeleken met de inhoud van registerpaar DE. Wordt hetzelfde regelnummer gevonden, dan set de routine op het einde de vlaggen Z en C, en in registerpaar BC komt het startadres van de programmaregel. Wordt niet hetzelfde, maar wel een hoger regelnummer gevonden, dan worden vlaggen NZ en NC gezet. Wordt het einde van het programmageheugen bereikt, dan wordt dat kenbaar gemaakt met de vlaggen Z en NC.

adres 42B2H

Deze routine wordt door de Hoofdlus van de Interpreter gebruikt om een tekstregel van tokens te voorzien. Bij de start bevat registerpaar HL het adres van het eerste karakter van de regel, in BUF. Op het einde van de routine staat de regel, van tokens voorzien, in KBUF. Registerpaar BC bevat de lengte van de regel en registerpaar HL het startadres ervan.

Met uitzondering van tekst die komt na "REM", "CALL" of "DATA", wordt elke reeks karakters die een keyword vormen , door het overeenkomstige token vervangen. Kleine letters worden in hoofdletters omgezet, om de vergelijking met keywords mogelijk te maken. Het "?"-teken wordt vervangen door het "PRINT"-token (91H). Het "'"-karakter wordt vervangen door ":" (3AH), een "REM"-token (8FH) en een "'"-token (E6H). Het "ELSE"-token (A1H) wordt voorafgegaan door een scheidingsteken tussen statements (3AH). Alle andere karakters in de tekst worden zonder meer gecopieerd, met uitzondering van kleine letters, die in hoofdletters worden omgezet. De tokens beneden 80H, de functie-tokens, kunnen niet rechtstreeks in KBUF geplaatst worden, omdat ze dan als gewone tekst zouden geïnterpreteerd worden. Daarom worden ze omgezet in de groep FFH, (token+80H).

Numerieke constanten worden eerst omgezet in één van de standaardtypes, in DAC (3299H). Daarna worden ze op een bepaalde manier opgeslagen, afhankelijk van het type en de orde van grootte. De regel is, dat zo weinig mogelijk geheugenruimte wordt gebruikt :

0BH LSB MSB .... octaal getal
0CH LSB MSB .... hexadecimaal getal
11H tot 1AH .... gehele getallen 0 tot 9
0FH LSB .... gehele getallen 10 tot 255
1CH LSB MSB .... gehele getallen 256 tot 32767
1DH EE CC CC CC .... enkele precisie
1FH EE CC CC CC CC CC CC CC .. dubbele precisie

Voor binaire getallen bestaat geen token : die getallen blijven als strings in de regel staan . Dit lijkt een erfenis te zijn van vroegere versies van Microsoft BASIC. Elk teken dat voor een getal staat wordt als een bewerkings-teken beschouwd, en als een apart teken opgeslagen. Tijdens het omzetten in tokens worden geen negatieve getallen geproduceerd. Doordat getallen in dubbele precisie zoveel ruimte innemen, kan het gebeuren dat een regel die veel dergelijke getallen bevat (bv. PRINT 1#,1#,1#, enz.) KBUF vol maakt. In dat geval wordt de foutmelding "Line buffer overflow" gegeven.

Elk getal dat volgt op één van de tokens in de tabel op 43B5H, wordt beschouwd als een regelnummer, en wordt met een apart token opgeslagen :

0DH LSB MSB ...... pointer
0EH LSB MSB ...... regelnummer

Tijdens het omzetten in tokens wordt het normale type (0EH) geproduceerd. Is een programma aan het lopen, dan worden deze regelnummers omgezet in het pointer-type (0DH), waardoor ze naar een adres verwijzen.

adres 43B5H

Hier staat een tabel met tokens, die door de voorga .de routine gebruikt wordt om te zoeken naar keywords die door een regelnummer gevolgd kunnen worden. Hieronder volgt de lijst van die keywords :

RESTORE	RUN
AUTO	LIST
RENUM	LLIST
DELETE	GOTO
RESUME	RETURN
ERL	THEN
ELSE	GOSUB

adres 4524H

Deze routine verwerkt het "FOR"-statement. Eerst wordt de lusvariabele gezocht, die zijn startwaarde krijgt via de routine die "LET" verwerkt (4880H). Het adres van de variabele wordt in registerpaar DE gezet. Het einde van het statement wordt gezocht (485BH) en dit adres wordt in ENDFOR gezet. Nu wordt de stack van de Z80 doorzocht (3FE6H) naar parameterblokken die dezelfde naam voor de lusvariabele gebruiken. Wordt zo een blok gevonden, dan wordt het adres in ENDFOR vergeleken met het adres dat in het gevonden parameterblok staat. Zijn die adressen gelijk, dan wordt het gevonden blok van de stack gehaald. Daardoor is het mogelijk, een lus onafgewerkt te laten, bijvoorbeeld door tijdens de verwerking van de lus met een "GOTO" terug naar het begin van de lus te springen.

De operand die de eindwaarde aangeeft, en de eventuele "STEP"-waarde worden nu geëvalueerd en in hetzelfde type omgezet als de lusvariabele. Na een controle of er nog voldoende ruimte op de stack is (625EH) wordt een parameterblok van vijfentwintig bytes op die stack gePUSHt. Dat blok bevat de volgende gegevens :

- 2 bytes ... ENDFOR-adres
- 2 bytes ... huidig regelnummer
- 8 bytes ... eindwaarde van lusvariabele
- 8 bytes ... STEP-waarde
- 1 byte ... type van lus
- 1 byte ... richting van STEP
- 2 bytes ... adres van lusvariabele
- 1 byte ... "FOR"-token (82H)

Dit parameterblok blijft op de stack staan, waar het door het "NEXT"-statement wordt geraadpleegd, tot de lus afgewerkt is. Dan wordt het van de stack gehaald. Het blok is altijd even groot, hoewel de volle acht bytes niet nodig zijn om, bij variabelen met een gehele waarde of met enkele precisie, de eindwaarde en de STEP-waarde te bevatten. In die genoemde gevallen worden de minst significante bytes gevuld met een willekeurige inhoud. Er dient opgemerkt,dat het soort rekenkundige bew.rking dat de "NEXT"-routine moet uitvoeren, en derhalve ook de snelheid waarmee de lus wordt afgewerkt, helemaal afhangt van het type van de lusvariabele, en niet van het type van de operands. Om met maximale snelheid te werken, moeten variabelen van het gehele type (bijvoorbeeld N%) worden gebruikt.

adres 4601H

Hier start de Verwerkingslus. Elke routine die een statement verwerkt, keert op het einde naar dit adres terug, zodat de Interpreter met de verwerking van het volgende statement kan beginnen. De momentele inhoud van de stack pointer van de Z80 wordt in SAVSTK gecopieerd ten behoeve van foutopvang-routines. Via de standaardroutine ISCNTC worden de "CTRL/STOP"-toetsen gecontroleerd. Ondertussen opgetreden interrupts worden verwerkt (6389H) en het adres in het programmageheugen waar de Verwerkingslus bezig was, wordt in SAVTXT gecopieerd. Dit adres staat tijdens de hele werking van de Interpreter in registerpaar HL.

Dan wordt het eerstvolgende karakter in de programmatekst onderzocht. Is het een scheidingsteken tussen statements (3AH), dan gaat de controle onmiddellijk over naar de uitvoeringsroutine (4640H). Is het karakter iets anders dan een eind-markeerder (00H), dan wordt de foutmelding "Syntax error" gegeven (405H), omdat er dan "illegale" tekst op het einde van een statement voorkwam. Registerpaar HL wordt verhoogd, tot het adres van de volgende regel, en het schakeladres wordt berekend. Is dat 0000H, dan is het programma beëindigd (4039H). Zoniet, wordt het regelnummer van deze regel in CURLIN gezet. Verschilt TRCFLG van nul, dan wordt het regelnummer op het scherm gezet (3412H), tussen vierkante haken. Daarna loopt de controle door in de uitvoeringsroutine.

adres 4640H

Op dit punt begint de Verwerkingslus met de uitvoering van een statement. Het adres van de start van de Verwerkingslus wordt op de stack van de Z80 gePUSHt (4601H), en via de standaardroutine CHRGTR wordt het eerste karakter van het nieuwe statement gelezen. Is het een "_" (5FH) dan wordt de controle overgedragen naar de routine die het "CALL"-statement verwerkt (55A7H). Is het kleiner dan 81H (het kleinste token dat een statement kan voorstellen) dan wordt de controle overgedragen aan de "LET"-routine (4880H). Is het karakter groter dan D8H (het grootste token dat een statement kan voorstellen), dan wordt bekeken of het één van de functietokens is die ook als statement gebruikt kunnen worden (51ADH). Ligt de code tussen die beide grenzen, dan wordt het adres van de juiste routine uit de tabel op adres 392EH gehaald, en op de stack van de Z80 gePUSHt. Daarna loopt deze routine door in de standaardroutine CHRGTR, die het volgende karakter uit het programmageheugen leest, waarna de controle overgedragen wordt aan de statement-verwerkers.

adres 4666H

---

naam	CHRGTR
in	HL - adres van karakter in programmageheugen
uit	A - volgende karakter in programmageheugen
wijzigt	AF, HL

Deze standaardroutine leest het volgende karakter in het programmageheugen. Registerpaar HL wordt opgehoogd, en het karakter in register A gezet. Als het een spatie, een "TAB"-code (09H) of een "LF"-code (0AH) is, wordt het overgeslagen. Als het een scheidingsteken tussen twee statements (3AH) of een eind-

markeerder (00H) is, eindigt de routine met vlaggen Z en NC. Is het een cijfer tussen "0" en "9" dan eindigt de routine met vlaggen NZ en C. Is het een ander karakter, met uitzondering van de tokens die voor de diverse types getallen staan, dan eindigt de routine met vlaggen NZ en NC. Als het karakter één van de genoemde tokens is, wordt het in CONSAV gezet en de operand wordt in CONLO gecopieerd. De code die het type aangeeft, wordt in CONTYP gezet en het adres van volgende karakter in het programma gaat naar CONTXT.

adres 46E8H

Deze routine wordt zowel door de Factor Evaluator gebruikt als tijdens het decoderen van de tokens, om een numerieke operand te lezen, wanneer de standaardroutine CHRGTR een numeriek token heeft gevonden. Eerst wordt het token uit CONSAV gelezen. Als het geen regelnummer is noch een pointer, dan wordt de operand uit CONLO naar DAC gecopieerd en de code die het type aangeeft, uit CONTYP naar VALTYP. Was het een regelnummer, dan wordt het in enkele precisie omgezet en in DAC gezet (3236H). Was het een pointer, dan wordt het oorspronkelijke regelnummer uit de regel gehaald waarnaar de pointer verwijst, in enkele precisie omgezet en in DAC geplaatst (3236H).

adres 4718H

Deze routine verwerkt het "DEFSTR"-statement. Register E wordt geladen met de code die een variabele van het string-type aangeeft (03H) en de controle gaat over op de algemene routine die types van variabelen definieert.

adres 471BH

Deze routine verwerkt het "DEFINT"-statement. Register E wordt geladen met de code die een variabele van het "geheel getal"-type aangeeft (02H) en de controle gaat over op de algemene routine die types van variabelen definieert.

adres 471EH

Deze routine verwerkt het "DEFSNG"-statement. Register E wordt geladen met de code die een variabele met enkele precisie aangeeft (04H) en de controle gaat over op de algemene routine die types van variabelen definieert.

adres 4721H

Deze routine verwerkt het "DEFDBL"-statement. Register E wordt geladen met de code die een variabele met dubbele precisie aangeeft (08H). Dan wordt het eerste karakter gecontroleerd, dat aangeeft voor welke groep van variabelen de definitie geldt (64A7H). Is dit geen hoofdletter, dan wordt de foutmelding "Syntax error" gegeven (4055H). Volgt na dit karakter een "-", dan wordt ook het tweede karakter gecontroleerd (64A7H). Het verschil tussen de twee karakters bepaalt hoeveel keer de code die met het opgegeven type overeenkomt, in DEFTBL ingevuld moet worden.

adres 4755H

Deze routine evalueert een operand en zet hem om in een geheel getal in registerpaar DE (520FH). Is de operand negatief, dan wordt de foutmelding "Illegal function call" gegeven.

adres 475FH

Deze routine wordt gebruikt door de routines die de statements verwerken die in de tabel op adres 43B5H staan. Ze leest een regelnummer uit de programmatekst, en zet dit om in een geheel getal zonder teken in registerpaar DE. Als het eerste gelezen karakter in de tekst een "." is (2EH) dan eindigt de routine met de inhoud van DOT in registerpaar DE. Als het een van de tokens is die een regelnummer voorafgaan (0DH of 0EH), bevat registerpaar DE op het einde van de routine de inhoud van CONLO. In de andere gevallen worden de opeenvolgende cijfers gelezen en bij het product opgeteld, met tussendoor de gepaste vermenigvuldigingen met tien, tot een niet-numeriek karakter wordt gelezen.

adres 479EH

Deze routine verwerkt het "RUN"-statement. Volgt er na dit statement geen regelnummer, dan wordt het hele werkgeheugen vrijgemaakt (62A9H) en de controle gaat over naar het begin van de Verwerkingslus, met in registerpaar HL het startadres van het programmageheugen. Volgde er wel een regelnummer, dan wordt het hele werkgeheugen vrijgemaakt (62A1H) en de controle wordt overgegeven aan de routine die het "GOTO"-statement verwerkt (47E7H). In het andere geval wordt aangenomen dat een bestandsnaam volgt, bijvoorbeeld in het statement : RUN "CAS:FILE", en de controle wordt overgedragen aan de routine die het "LOAD"-statement verwerkt (6B5BH).

adres 47B2H

Deze routine verwerkt het "GOSUB"-statement. Eerst wordt gecontroleerd of er nog voldoende ruimte op de stack is (625EH). Daarna wordt het regelnummer gelezen en in registerpaar DE gezet (4769H). Nu wordt een parameterblok van zeven bytes lang op de stack gePUSHt en de controle gaat over naar de routine die het "GOTO"-statement verwerkt (47EBH). Het parameterblok bevat de volgende informatie :

2 bytes ... adres van einde van het statement
2 bytes ... huidig regelnummer
2 bytes ... 0000H
1 byte ... "GOSUB"-token (8DH)

Dit parameterblok blijft op de stack staan tot een "RETURN"-statement wordt uitgevoerd. Dan wordt uit dit blok het adres bepaald waarheen de Interpreter terug moet keren, en het blok wordt gewist.

adres 47CFH

Deze routine wordt gebruikt door de interrupt-verwerker van de Verwerkingslus (6389H) om een parameterblok van hetzelfde type als bij een "GOSUB"-statement, op de stack van de Z80 te PUSHen.

Een interrupt-blok is op twee bytes na identiek aan een gewoon blok. De twee bytes die 0 bevatten bij een "GOSUB"-statement, bevatten in dit geval het adres van de plaats in TRPTBL die hoort bij het apparaat dat de interrupt veroorzaakte. Dit adres zal de routine die het "RETURN"-statement verwerkt, gebruiken om de interrupt-status van het apparaat aan te passen, na het beëindigen van een subroutine. Op het einde van deze routine gaat de controle over op de Verwerkingslus, om de programmaregel uit te voeren waarvan het adres in registerpaar DE staat.

adres 47E8H

Deze routine verwerkt het "GOTO"-statement. De operand die het regelnummer aangeeft, wordt gelezen (4769H) en in registerpaar HL gezet. Is het een pointer, dan wordt de controle onmiddellijk overgedragen aan de Verwerkingslus, om op de aangegeven plaats in het programmageheugen met de uitvoering te beginnen. Zoniet, wordt het regelnummer vergeleken met het huidige regelnummer, om de plaats te bepalen vanaf waar gezocht dient te worden. Is het opgegeven regelnummer groter, dan wordt gezocht vanaf het begin van de huidige regel (4298H). Is het kleiner, dan wordt er gezocht vanaf het begin van het programmageheugen (4295H). Wordt de opgegeven regel niet gevonden, dan wordt de foutmelding "Undefined line number" gegeven (481CH). In het andere geval wordt het regelnummer in de programmatekst vervangen door het startadres van die regel, en het token dat ervoor staat, wordt aangepast (5583H). Daarna wordt de controle overgedragen aan de Verwerkingslus, om de programmaregel waarnaar werd verwezen, uit te voeren.

adres 481CH

Deze routine produceert de foutmelding "Undefined line number".

adres 4821H

Deze routine verwerkt het "RETURN"-statement. In registerpaar DE wordt een fictief adres van een lusvariabele geladen, en de stack van de Z80 wordt doorlopen (3FE2H) op zoek naar het eerste parameterblok dat niet bij een "FOR"-lus hoort. Dit blok wordt van de stack gehaald. Wordt hier geen "GOSUB"-token gevonden (8DH), dan wordt de foutmelding "RETURN without GOSUB" gegeven.

De volgende twee bytes worden van de stack gehaald. Bevatten die iets anders dan nul, dan betekent dit dat het blok door een interruptroutine geproduceerd werd, en de tijdelijke "STOP"-toestand wordt ongedaan gemaakt (633EH). In het programmageheugen wordt dan gekeken of er na het "RETURN"-statement nog iets komt : dat moet dan een operand zijn die een regelnummer aangeeft, zodat de controle overgegeven wordt aan de "GOTO"-routine (47E8H). Is dat niet het geval, dan worden het regelnummer en het adres in het programmageheugen uit het blok gelezen, en de controle gaat over naar de Verwerkingslus.

adres 485BH

Deze routine verwerkt het "DATA"-statement. Er wordt over de programmaregel heen gelezen tot een scheidingsteken tussen twee statements (3AH) of een eind-markeerder (00H) wordt gevonden.

Deze routine verwerkt ook de "REM"- en "ELSE"-statements, wanneer ze op 485DH wordt aangeroepen, maar in deze gevallen wordt enkel de eind-markeerder (00H) als begrenzer aangenomen.

adres 4880H

Deze routine verwerkt het "LET"-statement. Eerst wordt de variabele opgezocht (5EA4H). Zijn adres wordt in TEMP opgeslagen en de operand geëvalueerd (4C64H). Indien nodig wordt dan het type van de operand aangepast, zodat dit hetzelfde is als het type van de variabele (517AH). Als de operand één van de drie numerieke types is, wordt hij gewoon uit DAC naar het adres van de variabele, in het variabelengeheugen, gecopieerd (2EF3H). Is de operand een string, dan wordt het adres van de inhoud van de string uit zijn signalement gelezen. Staat hij in KBUF, bijvoorbeeld wanneer het gaat om een expliciete string in een rechtstreeks ingetypt statement, dan wordt de inhoud eerst gecopieerd naar het stringgeheugen en een nieuw signalement opgesteld (6611H). Daarna wordt het signalement uit TEMPST gehaald (67EEH) en op het adres van de variabele, in het variabelengeheugen, gezet (2EF3H).

adres 48E4H

Deze routine verwerkt de statements : "ON ERROR", "ON DEVICE GOSUB" en "ON EXPRESSION". Wanneer het volgende karakter in het programma geen "ERROR"-token is (A6H), dan wordt de controle overgedragen aan de volgende routine, die "ON DEVICE GOSUB" en "ON EXPRESSION" verwerkt (490DH). Er wordt gecontroleerd of er een "GOTO"-token volgt (89H) en het daaropvolgende regelnummer wordt gelezen (4769H). Het programma wordt doorzocht naar het adres van de opgegeven programmaregel (4293H) en dit adres wordt in ONELIN gezet. Is het regelnummer niet nul, dan eindigt de routine hier. Is het regelnummer nul, dan wordt in ONEFLG gecontroleerd of er al een "error"-situatie bestaat (wat impliceert dat het statement voorkomt in een BASIC regel die deel uitmaakt van een fout-opvangroutine). Is dat zo, dan wordt de controle overgedragen aan de routine die fouten verwerkt (4096H), om een foutmelding te geven; zoniet, stopt de routine op de gewone manier.

adres 490DH

Deze routine verwerkt de "ON DEVICE GOSUB" en "ON EXPRESSION"-statements. Wanneer het volgende karakter in het programma geen token is dat naar een bepaald apparaat verwijst (7810H), wordt de controle overgedragen aan de "ON EXPRESSION"-routine (4943H). Eerst wordt gecontroleerd of een "GOSUB"-token aanwezig is (8DH). Dan worden om de beurt elk van de regelnummers die bij een bepaald apparaat horen, gelezen (4769H). Is dat regelnummer niet nul, dan wordt het programmageheugen doorzocht naar het adres van de opgegeven regel (4293H), en dit adres wordt op de plaats in TRPIBL gezet, die met het bedoelde apparaat overeenkomt (785CH). De routine stopt, wanneer geen regelnummers meer worden gelezen.

adres 4943H

Deze routine verwerkt het "ON EXPRESSION"-statement. De operand wordt geëvalueerd (521CH) en het daaropvolgende "GOSUB" of

"GOTO" - token (8DH of 89H) in register A gezet. Daarna wordt met behulp van de operand de programmatekst zolang doorlopen ,tot registerpaar HL het adres bevat van de vereiste operand die een regelnummer aangeeft. Daarna wordt de controle overgedragen aan de uitvoeringsroutine van de Verwerkingslus (4646H), om het "GOSUB" of "GOTO"-token te decoderen.

adres 495DH

Deze routine verwerkt het "RESUME"-statement. Vooraf wordt in ONEFLG gecontroleerd of er wel een foutconditie is. Is dat niet het geval, dan wordt de foutmelding "RESUME without error" gegeven (4064H). Wordt het statement gevolgd door een regelnummer, dat verschilt van nul, dan wordt de controle overgedragen aan de "GOTO"-routine (47EBH). Staat er een "NEXT"-token (83H), dan wordt uit ERRTXT en ERRLIN de plaats bepaald waar de fout optrad, het startadres van het volgende statement wordt opgezocht (485BH) en de routine stopt. Wordt geen regelnummer gevonden, of is het nul, dan wordt de plaats waar de fout optrad, uit ERRTXT en ERRLIN gehaald, en de routine stopt.

adres 49AAH

Deze routine verwerkt het "ERROR"-statement. De operand die er op volgt, wordt geëvalueerd, en in register E gezet (521CH). Als hij nul is, wordt de foutmelding "Illegal function call" gegeven (475AH). In het andere geval wordt de controle overgedragen aan de foutenverwerker (406FH).

adres 49B5H

Deze routine verwerkt het "AUTO"-statement. De eventuele operands die de startregel en de regelafstand aangeven, worden gelezen (475FH) en in AUTLIN en AUTINC gezet. Zijn er geen waarden gespecificeerd, dan wordt voor beide operands tien ingevuld. AUTFLG wordt op een van nul verschillende waarde gezet, het "RETURN"-adres van de Verwerkingslus wordt van de stack gehaald, en de controle wordt rechtstreeks aan de Hoofdlus overgedragen (4134H).

adres 49E5H

Deze routine verwerkt het "IF"-statement. De operand wordt geëvalueerd (4C64H), er wordt gecontroleerd of er een "GOTO"- of een "THEN"-token (89H of DAH) staat, en het teken van de operand wordt bepaald (2EA1H). Als de operand verschilt van nul ("waar" is) dan wordt de daaropvolgende programmatekst uitgevoerd door ofwel een direkte sprong naar de Verwerkingslus (4646H) of, als het om een regelnummer gaat, door de "GOTO"-routine (47E8H). Indien de operand nul is ("onwaar" is), wordt de tekst van het statement doorlopen (485BH) tot een "ELSE"-token (A1H) wordt gevonden waar geen "IF"-token tegenover staat, en de uitvoering wordt hervat.

adres 4A1DH

Deze routine verwerkt het "LPRINT"-statement. PRTFLG wordt op 01H gezet, waardoor de output naar de printer gestuurd wordt, en de controle gaat over op de "PRINT"-routine (4A29H).

adres 4A24H

Deze routine verwerkt het "PRINT"-statement. Eerst wordt bekeken of er na het token een buffer-nummer volgt en zo nodig wordt PTRFIL klaargezet om de output naar de gepaste In/Out-buffer te sturen (6D57H). Volgt geen tekst meer, dan wordt een "CR"-code en een "LF"-code gestuurd (7328H) en de routine stopt (4AFFH). Zo ja, worden de karakters één voor één bekeken. Staat er een "USING"-token (E4H), dan wordt de controle overgegeven aan de "PRINT USING"-routine (60B1H). Bij een ";"-karakter, wordt terug gesprongen naar het begin van de routine, om het volgende karakter te lezen (4A2EH). Wordt een "," gevonden, dan stuurt de routine voldoende spaties om de printpositie (in TTYPOS, LPTPOS of een controleblok van een In/Out-buffer) op een geheel veelvoud van veertien te brengen. Gaat de output naar het scherm, en is de printpositie groter dan of gelijk aan de inhoud van CLMLST, dan wordt in plaats daarvan een "CR"/"LF"-combinatie uitgevoerd (7328H). Datzelfde gebeurt indien de output naar een printer gaat, en de printpositie is groter dan, of gelijk aan, 238. Wordt een "SPC("-token gevonden (DFH), dan wordt de volgend operand geëvalueerd (521BH) en het berekende aantal spaties wordt geproduceerd. Wordt een "TAB("-token (DBH) gelezen, dan wordt de operand geëvalueerd (521BH) en de routine stuurt voldoende spaties om de huidige printpositie (die in TTYPOS, LPTPOS of een controleblok van een In/Out-buffer staat) op de gewenste positie te brengen.

Werd geen van de genoemde karakters gevonden, dan wordt het gegeven dat de tekst in dat geval bevat, geëvalueerd (4C64H). Is de operand een string, dan wordt hij gewoon doorgegeven (667BH). Is het een numerieke operand, dan wordt die eerst in tekst omgezet, in FBUFFR (3425H), en een passend stringsignalement gemaakt (6635H). Wordt de output naar een In/Out-buffer gestuurd, dan wordt de bekomen string afgedrukt (667BH). Is de output bedoeld voor het scherm of een printer, dan wordt eerst de laatste printpositie (in TTYPOS of LPTPOS) vergeleken met de lengte van een regel, en indien de output niet op de regel past, wordt een "CR"/"LF"-combinatie uitgevoerd (7328H). De maximale lengte van een regel op de printer is 255. De maximale lengte van een regel op het scherm wordt uit LINLEN gelezen. Nadat de string afgedrukt werd, wordt de controle overgedragen aan het beginpunt van de routine.

adres 4AFFH

Deze routine zet PTRFLG en PTRFIL op nul, waardoor de output van de Interpreter terug naar het scherm gaat.

adres 4B0EH

Deze routine verwerkt de drie statements : "LINE INPUT", "LINE INPUT #" en "LINE". Is het volgende karakter in het programma geen "INPUT"-token (85H), dan wordt de controle overgedragen aan de routine die het "LINE"-statement verwerkt (58A7H). Is het karakter na het "INPUT"-token, een "#" (23H), dan wordt de controle overgedragen aan de "LINE INPUT#"-routine (6D8FH).

Een eventuele string met hulptekst (de "prompt") wordt geëvalueerd en afgedrukt (4B7BH). De variabele wordt opgezocht (5EA4H) en er wordt gecontroleerd of het wel een string is

(3058H). Via de standaardroutine INLIN wordt een ingetypte tekstregel ingelezen. Is bij terugkeer uit de standaardroutine INLIN, vlag C geset (dat wil zeggen dat "CTRL/STOP" werd ingedrukt) dan wordt de controle overgedragen aan de "STOP"-routine (63FEH). Zoniet, wordt de input-string geanalyseerd en een passend signalement opgesteld (6638H). Daarna gaat de controle over op de "LET"-routine (4892H) waar de variabele wordt toegewezen. Het verdient vermelding dat, vooraleer de input gelezen wordt, het scherm niet in tekst-mode wordt gezet.

adres 4B3AH

Hier staat de tekst "?Redo from start", gevolgd door de combinatie "CR"/"LF" en een nul als stop-byte.

adres 4B4DH

Deze routine wordt gebruikt door de routine die de "READ" en "INPUT"-statements verwerkt, indien deze laatste er niet in slaagde om een bepaald gegeven in numerieke vorm om te zetten. Indien het om een "READ"-statement ging (wanneer FLGINP niet nul is), wordt de foutmelding "Syntax error" gegeven (404FH). In het andere geval wordt de tekst "?Redo from start" afgedrukt (6678H) en de controle springt terug naar de routine die het statement verwerkte.

adres 4B62H

Deze routine verwerkt het "INPUT#"-statement. Het nummer van de buffer wordt geëvalueerd en PIRFIL klaargezet om de input uit de aangegeven In/Out-buffer te halen (6D55H). Daarna gaat de controle over op de routine die tegelijk het "READ"- en "INPUT"-statement verwerkt (4B9BH).

adres 4B6CH

Deze routine verwerkt het "INPUT"-statement. Is het volgende karakter in het programma een "#", dan gaat de controle over op de "INPUT#"-routine (4B62H). Zoniet, wordt via TOTEXT het scherm in tekstmode gebracht, en een eventuele string met hulptekst wordt na analyse (6636H) op het scherm gebracht (6678H). Nu wordt een vraagteken afgedrukt, en via de standaardroutine QINLIN wordt een ingetypte regel tekst gelezen. Is bij terugkeer uit die routine vlag C geset ("CTRL/STOP"), dan gaat de controle over op de "STOP"-routine (63FEH). Wanneer het eerste karakter in BUF, nul is (lege input), dan wordt de routine beëindigd door naar het einde van het statement te springen (485AH). In het andere geval loopt de controle door in de gecombineerde "READ"/"INPUT"-routine.

adres 4B9FH

Deze routine verwerkt het "READ"-statement. Een groot deel ervan wordt ook door "INPUT" en "INPUT#" gebruikt, zodat de struktuur van de routine nogal vreemd is. Elke variabele die in de programmatekst staat , wordt opgezocht (5EA4H). Voor elk ervan wordt het overeenkomstige gegeven gelezen en aan de variabele toegewezen door de "LET"-routine (4893H). De gegevens worden, wanneer de routine door "READ" wordt gebruikt, uit de programmatekst gelezen, met behulp van de oorspronkelijke inhoud

van DATPTR (4C40H), terwijl ze uit de tekstbuffer BUF gelezen worden indien de routine door een van de "INPUT" - statements wordt gebruikt.

Indien er tijdens een "READ"-bewerking, geen gegevens meer worden gevonden, wordt een "Out of DATA"-foutmelding gegeven. Indien dit gebeurt bij een "INPUT", worden twee vraagtekens afgedrukt en een volgende ingetypte regel gelezen, via de standaardroutine QINLIN. Gebeurt het tijdens een "INPUT#"-routine, dan wordt nog een regel tekst uit de juiste In/Out-buffer naar BUF gecopieerd (6D83H). Is tijdens de verwerking van een "INPUT" de lijst met variabelen afgewerkt, dan wordt de tekst "Extra ignored" afgedrukt (6678H) en de routine stopt (4AFFH). Bij een "INPUT#" wordt geen bericht gegeven. Wordt een "READ"-statement verwerkt, dan wordt DATPTR aangepast (63DEH) en de routine stopt. Kan een bepaald gegeven niet in numerieke vorm omgezet worden, om in een numerieke variabele te worden opgeslagen (3299H) dan gaat de controle over op de "?Redo from start"-routine (4B4DH).

adres 4C2FH

Hier staat de tekst "?Extra ignored", gevolgd door de "CR"/"LF"-combinatie en een nul als stop-byte.

adres 4C40H

Deze routine wordt door de "READ"-routine gebruikt om het volgende "DATA"-statement in de programmatekst te localiseren. Het adres vanaf waar gezocht moet worden, staat in registerpaar HL. Elk statement in het programmageheugen wordt onderzocht, tot een "DATA"-token (84H) wordt gevonden, waarna de routine stopt (4BD1H). Wordt bij het zoeken, het laatste schakeladres bereikt, dan wordt een "Out of DATA"-foutmelding gegeven. Tijdens het zoeken wordt elk regelnummer in DATLIN gezet, waar de foutenverwerker het eventueel kan ophalen.

adres 4C5FH

Deze routine controleert of het volgende karakter in het programmageheugen het "="-token (EFH) is, waarna ze overgaat in de Expression Evaluator. Wordt de routine op adres 4C62H aangeroepen, dan wordt gecontroleerd of het volgende karakter een "(" is.

adres 4C64H

Deze routine evalueert een hele uitdrukking, en wordt in dit boek de "Expression Evaluator" genoemd. Bij de start ervan, bevat registerpaar HL het adres van het eerste karakter van de uitdrukking die geëvalueerd moet worden. Op het einde van de routine bevat registerpaar HL het adres van het karakter in het programmageheugen, dat volgt op de uitdrukking. Het resultaat van de evaluatie bevindt zich in DAC, en de code die het type van het resultaat aangeeft, in VALTYP. Is dat resultaat een string, dan bevindt het adres van het signalement zich op adres (DAC+2). Het signalement zelf, dat bestaat uit één byte die de lengte van de string bevat, en twee bytes met het adres van de string, staat ofwel in TEMPST ofwel in een string-variabele in het variabelengeheugen.

Een uitdrukking is een reeks factoren (4DC7H), die verbonden zijn door operatoren (tekens die een bewerking aangeven), die elk een bepaalde graad van voorrang hebben. Om een uitdrukking correct te kunnen verwerken, moet, indien een volgende operator een hogere voorrang heeft dan degene die werd verwerkt, de mogelijkheid bestaan dat een tussenresultaat tijdelijk wordt opgeslagen en dat er met een nieuwe berekening wordt begonnen. Daartoe beschikt de Expression Evaluator over twee fundamentele bewerkingen : opslaan (STACK) en uitvoeren (APPLY). Een voorbeeld :

```
3+250\2^2*3^3+1,
```

Deze uitdrukking wordt uitgewerkt in deze volgorde :

```
sla op     : 3+             (\ volgt)
sla op     : 250\           (^ volgt)
voer uit   : 2^2 = 4        (* volgt)
sla op     : 4*             (^ volgt)
voer uit   : 3^3 = 27       (+ volgt)
voer uit   : 4*27 = 108     (+ volgt)
voer uit   : 250\108 = 2    (+ volgt)
voer uit   : 3+2 = 5        (+ volgt)
voer uit   : 5+1 = 6        (, volgt)
```

De evaluatie stopt, wanneer de volgende operator een niveau van voorrang heeft dat gelijk is aan of lager is dan dat bij het begin, en de stack leeg is. Het teken dat de uitdrukking afsluit, een komma in het gegeven voorbeeld, krijgt een voorrang 0 toebedeeld, en stopt daardoor altijd de evaluatie. Normaal start de Expression Evaluator altijd met een voorrangsniveau nul, maar indien de routine op adres 4C67H wordt aangeroepen kan via register D een andere waarde ingesteld worden. Van deze mogelijkheid wordt door de Factor Evaluator gebruik gemaakt, om het bereik van de evaluatie te beperken, bij monadische negatie en bij de "NOT"-operatie.

adres 4D22H

Deze routine wordt door de Expression Evaluator gebruikt om een wiskundige bewerking (+, -, *, /, ^) uit te voeren op een paar numerieke operands. Er zijn aparte routines voor de relationele operatoren (4F57H) en de logische operatoren (4F78H). De eerste operand, de code die zijn type aangeeft, en het token van de operator, worden op de stack van de Z80 verwacht. De tweede operand en zijn type-code staan respectievelijk in DAC en in VALTYP . Eerst worden de types van beide operands vergeleken. Zijn die verschillend, dan wordt de operand met de kleinste precisie omgezet in hetzelfde type als de andere. Vervolgens worden de twee operands op de plaatsen gezet, waar ze moeten staan voor de wiskundige routines. Voor gehele getallen, wordt de eerste operand in registerpaar DE gezet en de tweede in registerpaar HL. Van twee operands met enkele precisie wordt de eerste in de registers C, B, E en D gezet, de tweede in DAC. Gaat het om twee operands met dubbele precisie, dan wordt de eerste in DAC en de tweede in ARG gezet. Dan wordt met behulp van het token van de operator uit een van de tabellen op 3D51H, 3D5DH of 3D69H, al naargelang het type, het gepaste adres gehaald, en de controle wordt aan de routine op dat adres overgedragen.

adres 4DB8H

Deze routine wordt door de Expression Evaluator gebruikt om twee gehele operands te delen. De eerste wordt in registerpaar DE verwacht, de tweede in registerpaar HL. Het resultaat wordt in DAC gezet. Beide operands worden omgezet in enkele precisie (2FCBH) en de controle gaat over op de routine die operands in enkele precisie deelt (3265H).

adres 4DC7H

Deze routine evalueert een factor van een uitdrukking, en wordt in dit boek de "Factor Evaluator" genoemd. Bij de start moet registerpaar HL het adres bevatten van het karakter dat staat vóór de factor die geëvalueerd moet worden. Op het einde van de routine bevat registerpaar HL het adres van het karakter dat volgt op de factor. Het resultaat staat dan in DAC en het type ervan wordt aangegeven door de inhoud van VALTYP. Een factor kan één van volgende gegevens zijn :

1) een numerieke constante of een string-constante
2) een numerieke variabele of een string-variabele
3) een functie
4) een monadische operator (+-NOT)
5) een uitdrukking tussen haakjes

Via de standaardroutine CHRGTR wordt het eerste karakter uit de programmatekst gelezen. Is het een karakter dat het einde van een statement aangeeft, dan wordt de foutmelding "Missing operand" gegeven (406AH).

Is het een ASCII-cijfer, dan wordt het in één van de standaard numerieke types omgezet, en in DAC geplaatst (3299H).

Is het een hoofdletter (64A8H), dan is het een variabele : de inhoud daarvan wordt opgezocht (4E9BH).

Gaat het om een token dat een getal voorafgaat, dan wordt het getal uit CONLO naar DAC gecopieerd (46B8H).

Is het één van de functie-tokens die door FFH worden voorafgegaan (zie de tabel op adres 39DEH), dan wordt het gedecodeerd, waarna de controle overgedragen wordt aan de overeenkomstige functie-routine (4EFCH).

Is het een monadische "+", dan wordt die gewoon overgeslagen. Enkel de monadische "-" (4E8DH) en "NOT" (4F63H) vergen een bewerking.

Is het onderzochte karakter een aanhalingsteken, dan wordt de daaropvolgende stringtekst geanalyseerd, en wordt een signalement opgesteld (6636H).

Is het karakter een "&", dan is de factor een niet-decimale numerieke constante : die wordt omgezet in een van de standaard numerieke types, en in DAC gezet (4EB8H).

Was de factor geen van de voorgaande, dan moet het een uitdrukking tussen haakjes zijn (4E87H), zoniet wordt de foutmelding "Syntax error" gegeven.

De routine controleert rechtstreeks of één van de onderstaande functie-tokens voorkomen, en geeft - indien dat zo is - de controle over aan het bijbehorende adres :

ERR	4DFDH	ATTR$	7C43H
ERL	4E0BH	VARPTR	4E41H
POINT	5803H	USR	4FD5H
TIME	7900H	INSTR	68EBH
SPRITE ...	7A84H	INKEY$	7347H
VDP	7B47H	STRING$ ...	6829H
BASE	7BCBH	INPUT$	6C87H
PLAY	791BH	CSRLIN	790AH
DSKI$	7C3EH	FN	5040H

adres 4DFDH

Deze routine wordt door de Factor Evaluator gebruikt om de "ERR"-functie uit te voeren. De inhoud van ERRFLG wordt, als een geheel getal, in DAC gezet (4FCFH).

adres 4E0BH

Deze routine wordt door de Factor Evaluator gebruikt om de "ERL"-functie uit te voeren. De inhoud van ERRLIN wordt, als een getal met enkele precisie, in DAC gecopieerd (3236H).

adres 4E41H

Deze routine wordt door de Factor Evaluator gebruikt om de "VARPTR"-functie uit te voeren. Staat er een "#" na het functie-token, dan wordt het buffer-nummer geëvalueerd (521BH), het controleblok van de In/Out-buffer gelocaliseerd (6A6DH) en het adres ervan als een geheel getal in DAC gezet (2F99H). In het andere geval wordt de variabele opgezocht (5F5DH) en zijn adres als een geheel getal in DAC gezet (2F99H).

adres 4E8DH

Deze routine wordt door de Factor Evaluator gebruikt om de monadische "-" uit te voeren. Register D wordt ingevuld met een voorrangsniveau van 7DH, de factor wordt geëvalueerd (4C67H) en het resultaat wordt negatief gemaakt (2E86H).

adres 4E9BH

Deze routine wordt door de Factor Evaluator gebruikt om de momentele waarde van een variabele op te zoeken. Eerst wordt de variabele gelocaliseerd (5EA4H). Gaat het om een stringvariabele, dan wordt zijn adres in DAC gezet, als verwijzing naar het signalement. In het andere geval wordt de inhoud van de variabele in DAC gecopieerd (2F08H).

adres 4EA9H

Deze routine laadt register A met het karakter waarvan het adres in registerpaar HL staat. Is het een kleine letter, dan wordt die omgezet in een hoofdletter.

adres 4EB8H

Deze routine wordt door de Factor Evaluator en de routine voor numerieke input (3299H) gebruikt om een getal dat wordt voorafgegaan door een "&", in een geheel getal in DAC om te zetten. Wordt een toegelaten karakter gelezen, dan wordt het product met 2, 8 of 16 vermenigvuldigd, afhankelijk van het karakter dat op het "&"-teken volgde, en de waarde van het gevonden teken wordt erbij opgeteld. Wordt het product te groot, dan wordt de foutmelding "Overflow" gegeven (4067H). De routine stopt, als een niet toegelaten karakter wordt gelezen.

adres 4EFCH

Deze routine wordt door de Factor Evaluator gebruikt om functie-tokens die door FFH worden voorafgegaan, te verwerken.

Is het token "LEFT$", "RIGHT$" of "MID$", dan wordt de volgende operand, die een string moet zijn, geëvalueerd (4C62H), het adres van het signalement op de stack van de Z80 gePUSHt, de numerieke operand die volgt, eveneens geëvalueerd (521CH) en op de stack gezet.

In de andere gevallen wordt de operand van de functie, die tussen haakjes staat, geëvalueerd (4E87H) en vervolgens in dubbele precisie omgezet (303AH) indien het token "SQR", "RND", "SIN","LOG","EXP","COS","TAN" of "ATN" voorstelde. Dan wordt met behulp van het token, het passende adres gelezen uit de tabel op adres 39DEH, en de controle wordt overgedragen aan de routine die de functies verwerkt.

adres 4F47H

Deze routine wordt gebruikt door de routine die numerieke input omzet (3299H) bij het zoeken naar een "+" of "-" karakter of token. Op het einde van de routine bevat register D 00H, indien het resultaat positief was, en FFH indien het negatief was.

adres 4F57H

Deze routine wordt door de Expression Evaluator gebruikt om een relatie uit te werken tussen twee operands. Zijn het numerieke operands, dan roept de Expression Evaluator eerst de wiskundige routine (4D22H) op, om de algemene relatiebewerkingen uit te voeren. Zijn de operands strings, dan wordt eerst de routine aangeroepen die strings vergelijkt (65C8H). Op dit punt bevat register A, samen met het vlagregister, het resultaat :

```
operand 1 = operand 2 ... A = 00H, vlag Z, NC
operand 1 < operand 2 ... A = 01H, vlag NZ, NC
operand 1 > operand 2 ... A = FFH, vlag NZ, C
```

De Expression Evaluator zorgt er ook voor dat het bitmasker met de oorspronkelijke operators op de stack van de Z80 staat. Dit masker bevat een "1"-bit op de plaats van de bewerking die uitgevoerd moet worden : 00000<=>. Dit masker wordt vergeleken met het resultaat van de relatie. De vergelijking geeft nul als resultaat aan, indien aan geen van de voorwaarden voldaan wordt. Dit wordt in DAC gezet als een geheel getal (2E9AH) : -1 voor "waar" of 0 voor "onwaar".

adres 4F63H

Deze routine wordt door de Factor Evaluator gebruikt om de monadische "NOT"- bewerking uit te voeren. In register D wordt een initieel voorrangsniveau van 5AH geladen, de uitdrukking wordt geëvalueerd (4C67H), en in een geheel getal omgezet (2FA8H). Daarna wordt het omgekeerd en in DAC teruggezet.

adres 4F78H

Deze routine wordt door de Expression Evaluator gebruikt om een logische bewerking ("OR", "AND", "XOR", "EQV", "IMP") of de "MOD" of "\"-bewerking uit te voeren op twee numerieke operands. De eerste operand, die al in een geheel getal werd omgezet, wordt op de stack van de Z80 verwacht, de tweede in DAC. Het token van de operator (eigenlijk het voorrangsniveau ervan) wordt in register B verwacht. Eerst wordt de tweede operand in een geheel getal omgezet (2F8AH) en dan wordt de operator onderzocht. Voor "MOD" en "\" zijn er afzonderlijke routines (323AH en 31E6H respectievelijk). De logische bewerkingen worden in de routine zelf uitgevoerd, door middel van de overeenkomstige instructies van de Z80, die op registerparen DE en HL worden toegepast. Het resultaat wordt als een geheel getal in DAC gezet (2F99H).

adres 4FC7H

Deze routine wordt door de Factor Evaluator gebruikt om de "LPOS"-functie toe te passen op een operand in DAC. De inhoud van LPTPOS wordt als een geheel getal in DAC gezet (4FCFH).

adres 4FCCH

Deze routine wordt door de Factor Evaluator gebruikt om de "POS"-functie toe te passen op een operand in DAC. De inhoud van TTYPOS wordt als een geheel getal in DAC gezet (2F99H).

adres 4FD5H

Deze routine wordt door de Factor Evaluator gebruikt om de "USR"-functie uit te voeren. Het getal tussen haakjes, dat dit token volgt, wordt rechtstreeks uit de tekst gehaald : het kan geen uitdrukking zijn. Het bijbehorende adres wordt uit USRTAB gelezen (4FF4H). De dan volgende operand tussen haakjes wordt geëvalueerd (4E87H) en in DAC gezet, als de doorgegeven parameter. Is de parameter een string, dan wordt de plaats waar hij staat, leeggemaakt (67D3H). De plaats in het programmageheugen waar de Interpreter staat, wordt op de stack van de Z80 gePUSHt, gevolgd door het "RET"- adres 3297H : de routine op dat adres zorgt ervoor dat de Interpreter op het correcte adres verderwerkt, na het beëindigen van de "USR"-functie. Nu wordt de controle overgedragen naar het opgegeven "USR"-adres, met in registerpaar HL het adres van het eerste byte in DAC, en in register A de code die het type van de operand aangeeft, uit VALTYP. Voor een string-parameter wordt daarenboven nog het adres van het signalement uit DAC gelezen en in registerpaar DE gezet.
De "USR"-routine kan elk register van de Z80 gebruiken, maar moet ervoor zorgen dat de Stack Pointer van de Z80 op het einde dezelfde inhoud heeft als bij het begin. De routine moet

eindigen met een "RET"-instructie. De interrupts mogen buiten werking gelaten worden, als dat nodig is, aangezien de Verwerkingslus ze toch weer in werking stelt. Een numerieke parameter, die de Interpreter moet verwerken, moet in DAC geplaatst worden. Normaal gezien moet die van hetzelfde numerieke type zijn als de parameter die vanuit BASIC werd doorgegeven. Wanneer evenwel VALTYP aangepast wordt door de routine, zal de Interpreter elk type van numerieke variabele accepteren.

Het is moeilijker om een string aan de Interpreter door te geven. Het gebruik van dezelfde methode als de stringfuncties van de Factor Evaluator, namelijk de string copiëren naar het stringgeheugen en een nieuw signalement in TEMPST zetten, is ingewikkeld en kan door wijzigingen in het MSX-systeem onbruikbaar worden. Een eenvoudiger en betrouwbaarder middel is, de meegebrachte parameter gebruiken om de ruimte te maken voor het resultaat. De parameter mag dan geen expliciete string zijn, omdat dan de tekst van het programma gewijzigd zou moeten worden. Er moet gebruik gemaakt worden van een impliciete parameter. Toch moet ook hierbij voorzichtigheid in acht genomen worden. De indruk kan gemakkelijk gewekt worden, dat de Interpreter het resultaat (de string) accepteert, terwijl dat helemaal niet zo is. Dat wordt door het volgende voorbeeld aangetoond. Dat doet niets anders dan de aangebrachte parameter terug naar BASIC brengen :

```
10 POKE &H9000,&HC9
20 DEFUSR=&H9000
30 A$=USR(STRING$(12,"!"))
40 PRINT A$
50 B$=STRING$(9,"X")
60 PRINT A$
```

Eerst lijkt het erop (regel 40) dat de aangebrachte string correct aan A$ werd toegewezen. Op regel 60 wordt het evenwel duidelijk dat de inhoud van A$ verknoeid werd door de toewijzing van B$ op regel 50. De reden daarvoor is, dat de tijdelijke opslagruimte die aan de aangebrachte parameter werd toegekend, uit het stringgeheugen werd teruggevorderd, voordat de "USR"-routine aangeroepen werd. Die ruimte werd daarna gebruikt om de inhoud van B$ in op te slaan, waardoor de inhoud van A$ werd overschreven. Dit kan vermeden worden door vooraf de parameter aan een variabele toe te wijzen, en dan die variabele als parameter door te geven:

```
10 A$ = STRING$(12,"!")
20 A$ = USR(A$)
```

Op regel 10 worden twaalf bytes van het stringgeheugen permanent aan A$ toegewezen. Bij de start van de "USR"-routine zal het signalement, waarvan het adres in registerpaar DE staat, het startadres bevatten van die twaalf bytes, waar het resultaat in opgeslagen moet worden. Indien het resultaat dat naar BASIC teruggebracht wordt, korter is dan de doorgegeven parameter, dan kan het byte in het signalement dat de lengte aangeeft, zonder enig neveneffekt gewijzigd worden. Voor meer details over de opslag van strings, verwijzen we naar de "Garbage Collector" (66B6H).

Het dient opgemerkt te worden, dat een "CLEAR"-commando niet strikt noodzakelijk is, voordat een machinetaalprogramma in het geheugen wordt geladen. Het gebied tussen het bovenste byte van het arraygeheugen en de onderkant van de stack van de Z80 wordt nooit door de Interpreter gebruikt. Op die plaats kan dus een machinetaalprogramma geladen worden, op voorwaarde dat de twee genoemde geheugenblokken er niet overheen bewegen.

adres 500EH

Deze routine verwerkt het "DEFUSR"-statement. Het getal dat erop volgt, wordt rechtstreeks uit de programmatekst gelezen en kan derhalve geen uitdrukking zijn. De bijpassende plaats in USRTAB wordt opgezocht (4FF4H), en na evaluatie van de operand die het adres bevat (542FH), wordt deze in USRTAB geplaatst.

adres 501DH

Deze routine verwerkt de "DEF"-statements. Als het volgende karakter een "USR"-token is (DDH), dan wordt de controle overgedragen aan de "DEFUSR"-routine (500EH). Anders wordt een "FN"-token gezocht (DEH). De variabele die de functie benoemt, wordt opgezocht (51A1H) en - na een controle of de Interpreter wel degelijk een programma aan het uitvoeren is - het huidige adres waar de Interpreter staat, wordt op die plaats gezet. Vervolgens worden alle variabelen uit de groep formele parameters om beurten gelocaliseerd (5EA4H), louter met de bedoeling om de variabelen alvast toe te wijzen. Hier eindigt de routine : de rest van het statement wordt overgeslagen (485BH) aangezien op dit ogenblik de functie niet uitgevoerd wordt.

adres 5040H

Deze routine wordt door de Factor Evaluator gebruikt om de "FN"-functie uit te voeren. De variabele die de functie benoemt wordt eerst gelocaliseerd (51A1H) om het adres te vinden van de functie-omschrijving in het programma. Alle formele variabelen uit die omschrijving worden op hun beurt gezocht (5EA4H) en hun adres wordt op de stack van de Z80 gePUSHt. Elke keer één van die variabelen gevonden wordt, wordt de overeenkomstige èchte parameter geëvalueerd (4C64H) en er direct onder op de stack gezet. Indien nodig wordt het type van de eigenlijke parameter omgezet in hetzelfde type als van de formele parameter (517AH).

Wanneer beide groepen afgewerkt zijn, worden één na één de adressen van de formele variabelen en de parameters van de stack gehaald. Elke variabele wordt gecopieerd uit het variabelen-geheugen naar PARM2, maar wel met de waarde van de eigenlijke parameter. Het dient vermeld dat, aangezien PARM2 maar honderd bytes lang is, slechts negen parameters in dubbele precisie zijn toegelaten. Zijn alle echte parameters gecopieerd in PARM2, dan wordt de hele inhoud van PARM1 (het huidige parametergebied) op de stack van de Z80 gePUSHt, en wordt PARM2 naar PARM1 gecopieerd (518EH). Registerpaar HL wordt geladen met het startadres van de omschrijving van de functie in het programma-geheugen, en de uitdrukking wordt geëvalueerd (4C5FH). De vroegere inhoud van PARM1 wordt van de stack gehaald, en terug op zijn vroegere plaats gezet. Tenslotte wordt het resultaat van de evaluatie, indien nodig, omgezet in hetzelfde type als dat van de variabele die de functie benoemt (517AH).

Het verschil tussen een zelf gedefinieerde functie en een gewone uitdrukking is alleen dat een functie een eigen groep van locale variabelen ter beschikking heeft. Die worden in PARM1 gecreëerd op het ogenblik dat de functie gebruikt wordt, en verdwijnen weer daarna. Wanneer de Expression Evaluator een variabele begint te zoeken, gaat hij in het variabelengeheugen kijken. Indien evenwel NOFUNS van nul verschilt (dat wil zeggen dat er tenminste één zelfgedefinieerde functie in werking is), dan kijkt de Expression Evaluator eerst in PARM1 en pas wanneer hij daar niet vindt wat hij zoekt, zal hij het globale variabelengeheugen doorzoeken. Het gebruik van een locaal variabelengeheugen, specifiek voor elke keer dat een functie wordt gebruikt, houdt in dat dezelfde namen kunnen gebruikt worden, zonder het risiko te lopen dat ze elkaar of de globale variabelen overschrijven.

Het dient opgemerkt te worden dat een functie die door de gebruiker werd gedefinieerd, trager wordt verwerkt dan dezelfde uitdrukking op de programmaregel die de functie aanroept, of zelfs trager dan een subroutine. Het vinden van de variabele die de functie benoemt, plus het veelvuldig gebruik van de stack, kan een aanzienlijke tijd in beslag nemen.

adres 5189H

Deze routine verplaatst een geheugenblok vertrekkende vanaf het adres dat in registerpaar DE staat naar het adres dat in registerpaar HL staat, met een lengte van BC bytes.

adres 5193H

Deze routine produceert de foutmelding "Illegal direct", indien uit CURLIN blijkt dat de Interpreter geen programma aan het verwerken is.

adres 51A1H

Deze routine controleert of er een "FN"-token (DEH) in de tekst staat, en wijst dan de variabele toe die de functie benoemt (5EA9H). Deze variabelen verschillen van gewone variabelen, doordat bit 7 van het eerste karakter van de naam geset is.

adres 51ADH

Indien bij het uitvoeringspunt van de Verwerkingslus (4640H) aan het begin van een statement een token gevonden wordt dat groter is dan D8H, krijgt deze routine de controle. Is het token geen functie-token met FFH ervoor, dan wordt de foutmelding "Syntax error" gegeven (4055H). Is het token één van de groep die ook een statement kan zijn, dan wordt de controle aan de desbetreffende routine overgedragen, zoniet wordt de foutmelding "Syntax error" gegeven. De bedoelde tokens zijn : "MID$" (696EH), "STRIG" (77BFH) en "INTERVAL" (77B1H). Er is geen apart token voorzien voor "INTERVAL" : het "INT"-token (85H) wordt gebruikt, en de rest van de tekst wordt door de routine zelf gelezen.

adres   51C9H

Deze routine verwerkt het "WIDTH"-statement. De operand wordt geëvalueerd (521CH) en de grootte ervan onderzocht. Is hij nul of groter dan tweeëndertig of veertig, afhankelijk van de scherm-mode (in OLDSCR), dan wordt de foutmelding "Illegal function call" gegeven (475AH). Is de operand even groot als de momentele inhoud van LINLEN, dan eindigt de routine zonder meer. Zoniet, wordt de scherminhoud gewist door middel van een "FORMFEED"-code (ØCH) via de standaardroutine OUTDO, dit voor het geval het scherm kleiner gemaakt moet worden. Daarna wordt de operand in LINLEN gezet en in LINL32 of LINL4Ø, afhankelijk van de inhoud van OLDSCR. Dan wordt het scherm nogmaals gewist, dit voor het geval het breder werd gemaakt. Doordat de variabele die de regellengte bevat, via OLDSCR en niet middels SCRMOD wordt bepaald, is het mogelijk de schermbreedte te wijzigen terwijl het scherm in grafische mode of veelkleurenmode staat. In dat geval wordt de wijziging van kracht bij een terugkeer naar de Hoofdlus van de Interpreter of wanneer een "INPUT"-statement uitgevoerd moet worden.

adres   52ØEH

Deze routine evalueert de volgende uitdrukking in het programma (4C64H) en zet het resultaat om in een geheel getal (2F8AH) dat in registerpaar DE geladen wordt. De grootte en het teken worden gecontroleerd, en de routine stopt.

adres   521BH

Deze routine evalueert de volgende operand in het programma (4C64H) en zet het resultaat om in een geheel getal (5212H). Is de operand groter dan 255, dan wordt de foutmelding "Illegal function call" gegeven (475AH).

adres   5229H

Deze routine verwerkt het "LLIST"-statement. PRTFLG wordt met Ø1H geladen, om de output naar de printer de sturen, en de routine gaat over in de "LIST"-routine.

adres   522EH

Deze routine verwerkt het "LIST"-statement. Eventuele operands die start- en eind-regelnummer aangeven, worden uit de tekst gelezen en hun startadres in het programmageheugen wordt opgezocht (4279H). De programmaregels worden nu afgedrukt, tot ofwel het schakeladres ØØØØH ontdekt wordt, ofwel op de "CTRL/STOP"-toetsen wordt gedrukt, of het opgegeven eindregelnummer werd bereikt. Daarna gaat de controle over naar het "Ok"-punt van de Hoofdlus (411FH). Het afdrukken van een regel verloopt als volgt : het regelnummer wordt gegeven (3412H), de tokens worden gedecodeerd (5284H) en de regel zelf wordt afgedrukt (527BH), waarna een "CR"/"LF"-combinatie uitgevoerd wordt (7328H).

adres   5284H

Deze routine wordt door de "LIST"-routine gebruikt om een programmaregel met tokens om te zetten in een tekstregel. Bij de

start bevat registerpaar HL het adres van het eerste karakter van de gecodeerde regel (met tokens). Op het einde staat de tekstregel in BUF, met een byte Ø erna.

Elk gewoon token en elk token dat door FFH wordt voorafgegaan, wordt in het overeenkomstig keyword omgezet, gewoon door dat woord te zoeken in de tabel op adres 3A72H. Dit gebeurt niet, indien voordien een aanhalingsteken werd gelezen, of een "REM"-token of een "DATA"-token. Gewoonlijk volgt daarna hoe dan ook gewone tekst, maar de controle gebeurt toch, om te vermijden dat grafische karakters als tokens geïnterpreteerd zouden worden. De groep ":" (3AH),"REM" (8FH),"'" (E6H) wordt omgezet in het karakter "'" (27H). Het scheidingsteken tussen twee statements (3AH) wordt voor een "ELSE"-token (A1H) weggehaald.

Wordt één van de numerieke tokens gevonden, dan worden de waarde en het type ervan eerst van CONLO en CONTYP in DAC en VALTYP gecopieerd (46E8H). Dan wordt het getal door de omzettingsroutines voor decimale (3425H), octale (371EH) of hexadecimale getallen (3722H) in tekstvorm gezet in FBUFFR. Bij een octaal en hexadecimaal getal wordt een "&"-teken ervoor gezet en de letter "O" of "H", al naargelang. Enkel indien er geen cijfers na de komma volgen noch een exponent ("E" of "D") wordt na getallen met enkele precisie of dubbele precisie, een "!" of "#" geplaatst.

adres 53E2H

Deze routine verwerkt het "DELETE"-statement. Het nummer van de startregel en dat van de eindregel wordt gelezen, als die opgegeven werden. In het programmageheugen wordt het startadres opgezocht (4279H). Indien er in het programmageheugen adres-pointers staan, worden die terug in regelnummers omgezet (54EAH). De eindregel wordt opgezocht (4295H) en indien dit adres lager is dan dat van de startregel, wordt de foutmelding "Illegal function call" gegeven (475AH). Zoniet, wordt de tekst "Ok" afgedrukt (6678H). Het geheugenblok vanaf het einde van de eindregel tot aan het begin van het variabelengeheugen wordt verzet naar het begin van de startregel. VARTAB, ARYTAB, STREND worden ingevuld met het nieuwe, lagere adres waar het programma eindigt. Dan gaat de controle direct over naar het einde van de Hoofdlus (4237H), waar de overige pointers aangepast worden, en het programmageheugen van nieuwe schakeladressen wordt voorzien. De controle gaat niet naar het normale "Ok"-punt, zodat het scherm niet in normale tekstmode wordt gezet. Dat houdt in dat, indien het scherm in grafische mode of in veelkleurenmode staat wanneer een "DELETE" wordt uitgevoerd - wat inderdaad weinig waarschijnlijk is - het systeem op hol slaat.

adres 541CH

Deze routine wordt door de Factor Evaluator gebruikt om de "PEEK"-functie toe te passen op een operand in DAC. De operand die het adres bevat, wordt gecontroleerd (5439H), waarna het byte uit het geheugen wordt gelezen en als een geheel getal in DAC wordt gezet (4FCFH).

adres 5423H

Deze routine verwerkt het "POKE"-statement. De operand die het

adres bevat, wordt gecontroleerd (542FH), daarna de operand die het byte bevat dat gePOKEt moet worden (521CH), en dan wordt het byte in het geheugen gezet.

adres 5439H

Deze routine zet de numerieke operand in DAC om in een geheel getal in registerpaar HL. De operand moet binnen het bereik -32768 tot +65535 liggen, en is gewoonlijk een adres, zoals bij "POKE", "PEEK", "BLOAD", enzovoort. Eerst wordt via de standaardroutine GETYPR het type van de operand gecontroleerd. Als blijkt dat het al een geheel getal is, wordt het gewoon in registerpaar HL gezet (2F8AH). Wanneer de operand in enkele of dubbele precisie staat, wordt het teken bekeken. Is het negatief, dan wordt het getal omgezet in een geheel getal (2F8AH). Zoniet, wordt het in enkele precisie gezet (2FB2H) en wordt de grootte bekeken (2F21H). Is het groter dan 32767 en kleiner dan 65536, dan wordt er -65536 bij opgeteld (324EH) vooraleer het in een geheel getal wordt omgerekend (2F8AH).

adres 5468H

Deze routine verwerkt het "RENUM"-statement. Staat er een nieuwe startregel vermeld, dan wordt die gelezen (475FH), zoniet wordt een standaardwaarde van tien aangenomen. Staat er een regelnummer vanaf waar het hernummeren moet plaatsvinden, dan wordt ook dit gelezen (475FH), zoniet wordt de standaardwaarde nul aangenomen. Volgt er nog een getal, dat de regelafstand bepaalt, dan wordt ook dit tenslotte gelezen (4769H), zoniet wordt een standaardwaarde van tien aangehouden.

Nu wordt het programmageheugen doorzocht naar bestaande regelnummers, groter dan of gelijk aan het opgegeven nummer voor de nieuwe startregel (4295H) en de vroegere beginregel (4295H). Indien het nieuwe adres lager is dan het oude adres, wordt de foutmelding "Illegal function call" gegeven (475AH). Daarmee wordt vermeden dat hogere programmaregels hernummerd worden naar lagere bestaande regelnummers.

Nu wordt een "loze" hernummering uitgevoerd, waarbij gecontroleerd wordt of er geen regelnummers ontstaan, groter dan 65529. Deze eerste poging is noodzakelijk : indien die fout optrad tijdens het echte hernummeren, zou de programmatekst er nogal verward uitzien. Is de controle uitgevoerd, dan worden alle regelnummers in de programmatekst omgezet in adrespointers (54F6H). Dit is een slimme oplossing voor het probleem van de keywords die naar regelnummers verwijzen, bijvoorbeeld "GOTO 50", omdat de programmatekst door het hernummeren niet van plaats veranderd wordt. Te beginnen met het opgegeven regelnummer worden nu alle nummers vervangen door de nieuwe waarde. Wanneer het schakeladres 0000H bereikt is, worden alle adrespointers opnieuw in regelnummers omgezet (54F1H), en de controle wordt overgedragen naar het "Ok"-punt van de Hoofdlus (411EH).

adres 54F6H

Wordt deze routine aangeroepen op adres 54F6H, dan verandert ze in een programma elke operand die een regelnummer bevat, in een pointer. Wordt ze aangeroepen op adres 54F7H, dan voert ze de

omgekeerde bewerking uit: dan verandert ze elke pointer in een operand die een regelnummer bevat. Te beginnen met het startadres van het programmageheugen, wordt elke regel doorzocht naar het token voor een pointer (ØDH) of een regelnummer (ØEH), al naargelang. Moeten pointers in regelnummers worden veranderd, dan wordt de pointer vervangen door het nummer van de regel waarnaar de pointer verwijst, en het token vooraf wordt vervangen door ØEH. In het andere geval, wordt het programmageheugen doorlopen (4295H) op zoek naar de vermelde regel. Wordt de regel gevonden, dan wordt de operand die het regelnummer bevatte, vervangen door het adres van de regel, en het token vooraf wordt in ØDH veranderd. Wordt de regel niet gevonden, dan wordt de foutmelding "Undefined line NNNN in NNNN" gegeven (6678H) en de routine werkt verder. Er is een speciale controle op "ON ERROR GOTO Ø"-statements, om onnodige foutmeldingen te voorkomen. Er wordt evenwel niet gelet op het gelijkaardige statement "RESUME Ø". Dit kan wel een foutmelding opleveren, maar die heeft dan niets te betekenen.

adres 555AH

Hier staat de tekst "Undefined line", gevolgd door een byte met nul.

adres 558CH
----

naam SYNCHR
in HL - adres van het te controleren karakter
uit A - volgende karakter in programmatekst
wijzigt AF, HL

Deze standaardroutine vergelijkt een karakter in de programmatekst, waarvan het adres in registerpaar HL staat, met een referentie-karakter. Dit laatste wordt aangebracht als een byte direkt na de "CALL"- of "RST"-instructie, bijvoorbeeld :

```
RST 8
DEFB ","
```

Gaat de vergelijking niet op, dan wordt een "Syntax error" gemeld (4055H). Zijn beide karakters gelijk, dan wordt de controle overgedragen aan de standaardroutine CHRGTR, die het volgende karakter in het programma ophaalt (4666H).

adres 5597H
----

naam GETYPR
in n.v.t.
uit AF - type van een operand
wijzigt AF

Deze standaardroutine zoekt het type uit van de operand die momenteel geëvalueerd wordt, zoals bepaald in VALTYP. De mogelijke resultaten zijn :

geheel getal .... A - FFH, vlag M, NZ, C
string .... A - ØØH, vlag P, Z, C
enkele precisie .... A - Ø1H, vlag P, NZ, C
dubbele precisie ... A - Ø5H, vlag P, NZ, NC

adres 55A8H

Deze routine verwerkt het "CALL"-statement. Eerst wordt de volledige naam van het statement uit de programmatekst naar PROCNM gecopieerd, waarbij ongebruikte bytes op nul worden gezet. De volledige naam kan immers een string van vijftien karakters lang zijn, zonder aanhalingstekens ervoor of erna. Het einde ervan wordt aangegeven door een "(", een ":" of een stop-byte (00H). Daarna wordt bit 5 van elk byte in SLTATR getest, om te weten of er een uitbreidings-ROM aanwezig is, die statements kan verwerken. Wordt een dergelijke ROM gevonden, dan zet de routine de overeenkomstige positie om in een Slot-identificator in register A, en een startadres voor die ROM in register H (7E2AH). Het adres van de routine die de statements verwerkt, wordt uit de ROM-adressen vier en vijf gelezen (7E1AH) en in registerpaar IX geladen. De Slot-identificator wordt in het MSB van registerpaar IY geladen, en via de standaardroutine CALSLT wordt de statement-verwerker in de gevonden ROM aangeroepen.

De ROM controleert de naam van het statement, en set vlag C indien hij die naam niet herkent. Zo ja, wordt de nodige bewerking uitgevoerd. Wordt de naam niet gevonden, dan wordt SLTATR verder onderzocht, tot de tabel helemaal doorlopen is, waarna de foutmelding "Syntax error" gegeven wordt (4055H). Werd de naam wel gevonden, dan stopt deze routine hier.

adres 55F8H

Deze routine wordt gebruikt door de routine die de naam van randapparatuur (zoals GRP, CRT, LPT en CAS, de vertaler) ontleedt (6F15H), wanneer die een naam in de programmatekst niet herkent. Bij de start bevat registerpaar HL het adres van het eerste karakter van de naam, en register B de lengte van de naam. Eerst wordt de naam naar PROCNM gecopieerd, gevolgd door een byte nul. Bit 6 van elk byte in SLTATR wordt dan getest, om te zien of er een uitbreidings-ROM aanwezig is die randapparatuur stuurt. Wordt een dergelijke ROM gevonden, dan wordt de plaats in SLTATR vertaald in een Slot-identificator in register A en een startadres voor de ROM in register H (7E2AH). Het adres van de besturings-routine wordt uit bytes 6 en 7 van de ROM gelezen (7E1AH) en in registerpaar IX geladen. De slot-identificator wordt in het MSB van registerpaar IY geladen, de code voor een onbekend randapparaat (FFH) in register A en via de standaard-routine CALSLT wordt het besturingsblok van de ROM aangeroepen.

De ROM onderzoekt nu de naam van het randapparaat. Herkent hij die niet, dan wordt vlag C geset, in het andere geval geeft hij zijn eigen code, tussen nul en drie, door. Werd de naam niet gevonden, dan wordt SLTATR verder doorlopen tot de hele tabel gelezen werd, waarna de foutmelding "Bad file name" wordt gegeven (6E6BH). Werd de naam wêl gevonden, dan wordt de interne code van de ROM opgeteld bij de overeenkomstige positie in SLTATR, vermenigvuldigd met vier, om een globale code voor het randapparaat te verkrijgen. De uitgangscode die bij elke post in SLTATR hoort, staat hieronder in hexadecimaal. De kolommen en rijen worden met "SS" en "PS" aangegeven : dat zijn de overeenkomstige primaire en secundaire Slot-nummers. Elk Slot

beslaat vier bladzijden.

SS0				SS1				SS2				SS3				
00	04	08	0C	10	14	18	1C	20	24	28	2C	30	34	38	3C	PS0
40	44	48	4C	50	54	58	5C	60	64	68	6C	70	74	78	7C	PS1
80	84	88	8C	90	94	98	9C	A0	A4	A8	AC	B0	B4	B8	BC	PS2
C0	C4	C8	CC	D0	D4	D8	DC	E0	E4	E8	EC	F0	F4	F8	FC	PS3

Fig. 44 : codes van randapparaten

De globale code wordt door de Interpreter gebruikt, tot op het moment dat de ROM echt een bewerking moet uitvoeren met één of ander randapparaat. Dan wordt de code opnieuw omgezet in een Slot-identificator, een startadres en een interne code, om de toegang tot de ROM mogelijk te maken. Merk op dat de codes van 0 tot 8 gereserveerd zijn voor disk-drive benamingen, en die tussen FCH en FFH voor de standaard-apparatuur, zoals GRP, CRT, LPT en CAS. Zoals de MSX-hardware momenteel gestructureerd is, komen deze laatste codes overeen met fysiek onwaarschijnlijke ROM-configuraties; daardoor kan de Interpreter ze veilig gebruiken voor specifieke doeleinden.

adres 564AH

Deze routine wordt gebruikt door de In/Out-functiebeheerder (6F8FH), wanneer die een code voor een randapparaat leest die niet bij een van de standaard-aparaten behoort. Eerst wordt die code omgerekend naar een positie in SLTAIR, daarna naar een Slot-identificator in register A en een startadres voor de ROM in register H (7E2DH). Het adres van de besturingsroutines in ROM wordt uit bytes zes en zeven van de ROM gelezen (7E1AH) en in registerpaar IX geladen. De Slot-identificator wordt in het MSB van registerpaar IY geladen, de interne code van de ROM in DEVICE en daarna wordt het besturingsblok in de ROM via de standaardroutine CALSLT aangeroepen.

adres 566CH

Dit is het toegangsadres tot de ontledingsroutine van de macro-taal, dat door het "DRAW"-statement wordt gebruikt. Adres 56A2H wordt als toegangspunt gebruikt door de routine die het "PLAY"-statement verwerkt. De commando-string wordt geëvalueerd (4C64H) en de opslagruimte ervan vrijgemaakt (67D0H). Een eindblok met nullen wordt op de stack van de Z80 gePUSHt, de lengte en het adres van de inhoud van de string worden in MCLLEN en MCLPTR geladen, en de controle gaat over in de hoofdlus van de ontledings-routine.

adres 56A2H

Dit is de hoofdlus van de macro-ontleder. Hij dient om de commando-string, die bij een "DRAW"- of "PLAY"-statement hoort, te verwerken. Bij de start staat de lengte van de string in MCLLEN en het adres ervan in MCLPTR. Het adres van de

overeenkomstige tabel met commando's staat in MCLTAB. Deze tabellen (voor elk statement één) bevatten de letters die een geldig commando voorstellen, met de adressen van de bijbehorende routines waar de commando's worden verwerkt. De tabel die bij "DRAW" hoort, staat op adres 5D83H, de tabel voor "PLAY" op adres 752EH.

De hoofdlus leest eerst het volgende karakter uit de commandostring (56EEH). Zijn er geen karakters meer, dan wordt het volgende signalement van de stack gePOPt (568CH). Is die nul, dan eindigt de ontleding (5709H) indien MCLFLG aangeeft dat een "DRAW"-statement werd uitgevoerd. Zoniet, wordt de controle weer overgegeven aan de "PLAY"-routine (7494H).

Wordt wél een karakter gevonden, dan wordt de gebruikte tabel doorzocht om te controleren of dat karakter toegestaan is. Is dat niet het geval, dan wordt de foutmelding "Illegal function call" gegeven (475AH). Zo ja, wordt de gevonden plaats in de tabel verder onderzocht. Is bit 7 geset, dan kan er na dit commando nog een numerieke parameter volgen. Is die er, dan wordt hij gelezen en in registerpaar DE geladen (571CH). Zoniet, wordt een standaardwaarde van één aangenomen. Een "RETURN"-adres naar het beginpunt van de ontleder wordt op de stack van de Z80 geplaatst, en de controle wordt overgedragen aan de routine op het adres dat uit de tabel werd gelezen.

adres 56EEH

Deze routine wordt door de macro-ontleder gebruikt om een volgend karakter uit een commandostring te lezen. Bevat MCLLEN nul, dan stopt de routine met vlag Z, als teken dat er geen karakters meer te lezen zijn. In het andere geval wordt een karakter gelezen op het adres dat in MCLPTR staat, en in register A geladen. Is het een kleine letter, dan wordt hij omgezet in een hoofdletter. MCLPTR wordt opgehoogd, MCLLEN wordt met één verlaagd.

adres 570BH

Deze routine wordt door de macro-ontleder gebruikt om een ongewenst karakter terug in de commandostring te zetten. MCLLEN wordt opgehoogd, en MCLPTR met één verlaagd.

adres 5719H

Deze routine wordt door de macro-ontleder gebruikt om een numerieke parameter uit de commandostring te lezen. Het resultaat wordt als een geheel getal met een teken in registerpaar DE gezet. De parameter kan geen uitdrukking zijn. Het eerste karakter wordt onderzocht. Is het een "+", dan wordt het volgende karakter gelezen (5719H). Is het een "-", dan wordt een "RET"-adres naar de negatie-routine klaargezet (5795H) en het volgende karakter wordt gelezen (5719H). Is het eerste karakter een "=", dan wordt de waarde van de daaropvolgende variabele opgezocht (577AH). In de andere gevallen worden opeenvolgende karakters gelezen en het totaal van de binaire producten bijgehouden, tot een niet-numeriek karakter wordt gelezen.

adres 575AH

Deze routine wordt gebruikt door de macro-ontleder om de combinatie "=" en "X" te verwerken. De naam van de variabele wordt naar BUF gecopieerd tot het ";"-teken gevonden wordt, dat het einde aanduidt. Staan er meer dan negenendertig karakters voor dat eind-teken, dan wordt de foutmelding "Illegal function call" gegeven (475AH). In het andere geval wordt de controle overgedragen aan de variabelen-verwerker van de Factor Evaluator (4E9BH). De inhoud van de variabele wordt in DAC gezet.

adres 577AH

Deze routine wordt door de macro-ontleder gebruikt bij de verwerking van het "="-karakter in een commando-parameter. De waarde van de variabele wordt berekend (575AH), in een geheel getal omgezet (2F8AH) en in registerpaar DE gezet.

adres 5782H

Deze routine wordt door de macro-ontleder gebruikt bij het verwerken van het "X"-commando. De variabele wordt verwerkt (575AH) en de momentele waarde van MCLLEN en MCLPTR wordt op de stack gezet, na een controle of er voldoende plaats was (625EH). Daarna gaat de controle over naar het startpunt van de ontledingsroutine (5679H), om het signalement van de variabele te halen, en de nieuwe commandostring te verwerken.

adres 579CH

Deze routine wordt door diverse grafische statements gebruikt om een coördinatenpaar in de programmatekst te evalueren. Die coördinaten moeten tussen haakjes staan, met een komma tussen de operands. Wordt een coördinatenpaar voorafgegaan door een "STEP"-token (DCH) dan wordt de waarde van elke component opgeteld bij de overeenkomstige waarde van de huidige grafische coördinaten in GRPACX en GRPACY. In het andere geval worden de absolute waarden berekend. HET X-coördinaat wordt in GRPACX gezet, in GXPOS en in registerpaar BC. Het Y-coördinaat in GRPACY, GYPOS en registerpaar DE.

Deze routine heeft twee ingangspunten. Welk punt van de twee gebruikt wordt, hangt af van het feit of de gebruiker meer dan één coördinatenpaar verwacht. Het "LINE"-statement verwacht bijvoorbeeld twee paren, waarvan het eerste de meeste vormen kan aannemen. Het toegangsadres 579CH wordt gebruikt om het eerste coördinatenpaar op te halen, en de karakters "-" en "@" worden geaccepteerd als symbool voor de huidige grafische coördinaten. Het toegangsadres 57ABH wordt gebruikt om het tweede coördinatenpaar te lezen, daarbij moeten de operands wel uitdrukkelijk worden opgegeven.

adres 57E5H

Deze routine verwerkt het "PRESET"-statement. Uit BAKCLR wordt de momentele achtergrondkleur gelezen, en de controle loopt door in de "PSET"-routine.

adres 57EAH

Deze routine verwerkt het "PSET"-statement. Eerst wordt het coördinatenpaar geëvalueerd (57ABH). Dan wordt uit FORCLR de momentele voorgrondkleur gelezen en als standaard genomen om de inktkleur in te vullen (5850H). De huidige grafische coördinaten worden in een scherm-adres omgerekend via de standaardroutines SCALXY en MAPXYC, en de standaardroutine SETC vult de kleur in van de berekende pixel.

adres 5803H

Deze routine wordt door de Factor Evaluator gebruikt om de "POINT"-functie uit te werken. De momentele inhoud van CLOC, CMASK, GYPOS, GXPOS, GRPACY en GRPACX worden op de stack gezet, en de operand van het coördinatenpaar geëvalueerd (57ABH). Via de standaardroutines SCALXY, MAPXYC en READC wordt de kleur van de nieuwe pixel gelezen en als een geheel getal in DAC gezet (2F99H). De oude waarden van de coördinaten worden van de stack gehaald en terug in de variabelen gezet. Merk op, dat een waarde van -1 wordt gegeven indien de opgegeven coördinaten buiten het scherm lagen.

adres 5850H

Deze grafische routine wordt gebruikt om een facultatieve kleur-operand in de programmatekst te evalueren, en die tot gangbare inktkleur te maken. De schermmode wordt gecontroleerd (59BCH), en dan wordt de kleur-operand geëvalueerd (521CH) en in ATRBYT gezet. Wordt geen kleur opgegeven, dan wordt in de plaats daarvan de kleurcode die in register A staat, in ATRBYT gezet.

adres 5871H

Deze grafische routine zet het verschil tussen de inhoud van GXPOS en registerpaar BC, in registerpaar HL. Indien het resultaat negatief is (GXPOS < BC) dan wordt het teken omgekeerd om de absolute waarde te kennen, en vlag C wordt geset.

adres 5883H

Deze grafische routine zet het verschil tussen de inhoud van GYPOS en registerpaar DE in registerpaar HL. Indien het resultaat negatief is (GYPOS < DE) dan wordt het teken omgekeerd om de absolute waarde te kennen, en vlag C wordt geset.

adres 588EH

Deze grafische routine verwisselt de inhoud van GYPOS met de inhoud van registerpaar DE.

adres 5898H

Deze grafische routine verwisselt eerst de inhoud van GYPOS met de inhoud van registerpaar DE (588EH) en dan de inhoud van GXPOS met de inhoud van registerpaar BC. Indien de routine op adres 5898H wordt aangeroepen, wordt enkel de laatste bewerking uitgevoerd.

adres 58A7H

Deze routine verwerkt het "LINE"-statement. Het eerste coördinatenpaar (X1,Y1) wordt geëvalueerd (579CH) en in registerparen BC en DE gezet. Na een controle of het "-" token (F2H) voorkomt, wordt het tweede coördinatenpaar (X2,Y2) geëvalueerd (57ABH) en in GRPACX, GRPACY en GXPOS, GYPOS gezet. De inktkleur wordt ingevuld, en dan wordt gecontroleerd of er een "B" of "BF" optie volgt. In voorkomend geval wordt een "BOX" (5912H) of "BOXFILL" (58BFH) uitgevoerd of gewoon een lijn getrokken (58FCH). Geen van deze drie bewerkingen heeft een invloed op de huidige grafische coördinaten in GRPACX en GRPACY, die op het punt (X2,Y2) blijven staan.

adres 58BFH

Deze routine voert de "BOXFILL"-bewerking uit. Gegeven dat de berekende coördinatenparen, diagonaal liggende punten van de rechthoek definiëren, moeten twee grootheden afgeleid worden. De horizontale grootte van de rechthoek wordt berekend door het verschil te berekenen tussen X1 en X2 : dat levert het aantal pixels op dat per pixelrij geset moet worden. De vertikale grootte wordt berekend uit het verschil tussen Y1 en Y2 . Dat levert het aantal pixelrijen op. Te beginnen bij de schermpositie (X1,Y1), en via de standaardroutine DOWNC naar beneden toe werkend over het aantal berekende pixelrijen, wordt elke pixelrij over de berekende lengte door de standaardroutine NSETCX ingevuld.

adres 58FCH

Deze routine tekent een lijn. Op het einde (593CH) worden GXPOS en GYPOS teruggezet op de waarden X2 en Y2, die in GRPACX en GRPACY stonden.

adres 5912H

Deze routine tekent een rechthoek. Dit gebeurt door een lijn te trekken (58FCH) tussen elk van de vier hoekpunten. De coördinaten van elke hoek worden afgeleid van de oorspronkelijke operands, door de toepasselijke component in het paar om te wisselen. De volgorde bij het tekenen is deze :

1) X1,Y1 - X2,Y2
2) X1,Y1 - X2,Y1
3) X2,Y1 - X2,Y2
4) X1,Y1 - X1,Y2

adres 593CH

Deze routine trekt een lijn tussen de punten X1,Y1 in de registerparen BC en DE, en X2,Y2 in GXPOS en GYPOS. De werking van de hoofdlus van de tekenroutine wordt het beste aan de hand van een voorbeeld verduidelijkt, stel LINE(0,0)-(10,4). Vanaf het startpunt naar het einde van de lijn moeten tien horizontale stappen (X2-X1) en vier stappen naar beneden (Y2-Y1) berekend worden. De beste benadering van een rechte lijn wordt dus bereikt door twee en een half horizontale stappen te zetten voor

elke vertikale stap ((X2-X1)/(Y2-Y1)). Dit is in de praktijk onmogelijk, omdat er enkel gehele stappen gezet kunnen worden. Toch kan gemiddeld de juiste verhouding bereikt worden.

De methode die wordt toegepast, is de volgende : elke keer als een stap naar rechts wordt gezet, wordt het Y-verschil opgeteld bij een teller. Als die teller groter wordt dan het X-verschil, wordt de teller terug op zijn oorspronkelijke stand gezet, en de tekenroutine zet een stap naar beneden. Dit komt in feite neer op een gehele deling van de twee verschil-waarden. Soms wordt een stap naar beneden gezet na twee horizontale stappen en soms na drie. Het gemiddelde zal neerkomen op een stap naar beneden elke twee en een halve horizontale stap. Hieronder volgt een BASIC programma dat eerst een lijn tekent volgens dezelfde methode, met daarna een gewoon LINE-statement een paar pixels lager op het scherm, ter vergelijking.

```
10 SCREEN 0
20 INPUT"START X,Y";X1,Y1
30 INPUT"END X,Y";X2,Y2
40 SCREEN 2
50 X=X1:Y=Y1:L=X2-X1:S=Y2-Y1:CTR=L/2
60 PSET(X,Y)
70 CTR=CTR+S:IF CTR<L THEN 90
80 CTR=CTR-L:Y=Y+1
90 X=X+1:IF X<=X2 THEN 60
100 LINE (X1,Y1+5)-(X2,Y2+5)
110 GOTO 110
```

Het voorbeeld hierboven heeft drie beperkingen : de lijn moet naar beneden toe hellen, naar rechts, en de helling mag niet meer dan vijfenveertig graden bedragen (één stap naar beneden voor elke stap horizontaal).

De ROM-routine omzeilt de eerste beperking door de Y1 en Y2 coördinaten te vergelijken vooraleer ze te tekenen. Indien Y2 groter is dan, of gelijk aan Y1, dan worden beide coördinaten-paren omgewisseld, omdat in dat geval de lijn omhoog helt of horizontaal loopt. Nu helt de lijn neerwaarts. Ze wordt getekend vanaf het eindpunt naar het beginpunt toe.

De tweede beperking wordt omzeild door vooraf X1 en X2 te vergelijken om te bepalen naar welke kant de lijn helt. Als X2 groter is dan of gelijk aan X1, dan helt de lijn naar rechts en er wordt in MINUPD/MAXUPD een JP-instructie naar de standaard-routine RIGHTC geschreven (zie verderop), ten behoeve van de hoofdlus van de tekenroutine. In het andere geval wordt een JP naar de LEFTC standaardroutine op dezelfde plaats gezet.

De derde beperking wordt omzeild door het X-verschil met het Y-verschil te vergelijken, om de hellingshoek van de lijn te bepalen. Als X2-X1 kleiner is dan Y2-Y1, dan is de hellingshoek kleiner dan vijfenveertig graden horizontaal. De eenvoudige methode, zoals hoger vermeld voor het voorbeeld LINE(0,0)-(10,4) werkt niet voor hoeken groter dan vijfenveertig graden, omdat de steilste helling bereikt wordt indien er voor elke horizontale stap ook een vertikale stap wordt geteld. De methode zal wél werken indien de stap-richtingen omgewisseld worden. Zo zal LINE(0,0)-(4,10) één stap naar rechts vergen voor elke twee en een halve stappen omlaag. In MINUPD staat een JP-instructie naar

de standaardroutine die "normale" staprichtingen voor de tekenroutine verzorgt, en MAXUPD bevat een JP-instructie naar de standaardroutine die "schuine" staprichtingen verwerkt. Voor lichte hellingen zal MINUPD verwijzen naar de standaardroutine DOWNC en MAXUPD naar de standaardroutine LEFTC of RIGHTC. Voor steile hellingen zal MINUPD verwijzen naar de standaardroutine LEFTC of RIGHTC en MAXUPD naar de standaardroutine DOWNC. Bij steile hellingen moeten ook de waarden van de tellers verwisseld worden : nu moet het X-verschil bij de teller opgeteld worden en het Y-verschil gebruikt worden als de limiet voor de teller. De variabelen MINDEL en MAXDEL dienen om deze tellerwaarden bij te houden voor de tekenroutine. MINDEL bevat het kleinste van de twee verschillen, MAXDEL het grootste.

Het is interessant dat de referentieteller, CTR in het programma en registerpaar DE in de ROM, geladen wordt met de helft van het grootste verschil, in plaats van nul. Daardoor wordt de eerste "trap" in de lijn in twee stukken gedeeld, één bij het begin en één op het einde, waardoor het uitzicht van de lijn wordt verbeterd.

adres 59B4H

Deze grafische routine schuift de inhoud van het registerpaar DE één bit naar rechts.

adres 59BCH

Deze routine produceert de foutmelding "Illegal function call" (475AH) indien het scherm niet in grafische mode of veelkleuren-mode staat.

adres 59C5H

Deze routine verwerkt het "PAINT"-statement. Het coördinatenpaar van het startpunt wordt geëvalueerd (579CH), de inktkleur wordt ingevuld (584DH) en de facultatieve operand die de grenskleur aangeeft wordt geëvalueerd (521CH) en in BDRATR gezet. Er wordt gecontroleerd of het startpunt nog binnen het scherm ligt (5E91H). Dit punt wordt tot courante schermpositie gemaakt via de standaardroutine MAPXYC. Daarna wordt de afstand tot aan de rechter grenslijn gemeten (5ADCH) en als die nul is, stopt de routine. In het andere geval wordt de afstand tot aan de linker grenslijn gemeten (5AEDH) en de som van de twee afstanden in registerpaar DE gezet, als de breedte van het gebied. De courante positie wordt nu twee keer op de stack gezet (5ACEH), eerst met een eind-byte (00H) en dan met een vlag die aangeeft dat omlaag gewerkt moet worden (40H). Dan wordt de controle overgedragen aan de hoofdlus van de opvul-routine (5A26H) met in register B een vlag voor "opwaarts werken" (C0H).

adres 5A26H

Hier begint de hoofdlus van de "PAINT"-routine. De breedte van de zone die opgevuld moet worden, staat in registerpaar DE. Register B bevat de richting waarin gewerkt moet worden, omhoog of omlaag. De courante pixelpositie is de plaats naast de linker grenslijn. De routine zet een stap in vertikale richting, naar de volgende pixellijn, via de standaardroutines TUPC of TDOWNC waarna de afstand tot aan de rechter grenslijn wordt gemeten (5ADCH). Daarna wordt de afstand tot aan de linker grenslijn

gemeten en de pixellijn tussen die twee punten opgevuld (5AEDH). Wordt geen verandering vastgesteld in de positie van beide grenslijnen, dan gaat de controle terug naar het startpunt van de hoofdlus, om in dezelfde riching verder te werken. Wordt er een verandering vastgesteld, dan staat op dat punt een inham of een uitstulping en moet de juiste actie ondernomen worden.

Er zijn vier types van onregelmatigheden : rechtse of linkse inhammen, waarbij de overeenkomstige grenslijn naar binnen toe verandert, en rechtse of linkse uitstulpingen, waarbij de grenslijn naar buiten toe verandert. Hieronder staat een voorbeeld van elk type, met genummerde zones, die de volgorde aangeven waarin ze gevuld worden wanneer de routine naar boven toe werkt. Er wordt ook, voor de volledigheid, een secundaire uitstulping getoond :

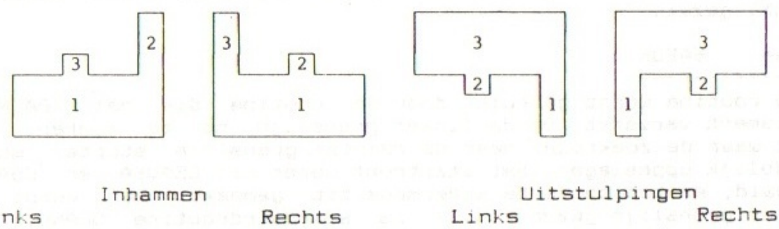


Fig. 45 : onregelmatigheden in de grenslijn van een figuur

Er bestaat een linkse uitstulping , wanneer de afstand tot aan de linker grenslijn van nul verschilt. Een rechtse uitstulping bestaat wanneer de momentele breedte van de zone groter is dan die van de vorige lijn. Wanneer de uitstulping smaller is dan twee pixels, wordt ze genegeerd. In het andere geval wordt de momentele positie (het punt links onder de zone 3 in fig. 45) op de stack gezet (5AC2H). Daarna wordt de werkrichting omgekeerd, en het opvullen begint weer links bovenaan de uitstulping.

Er bestaat een rechtse inham indien de momentele breedte van de zone kleiner is dan die van de vorige lijn. Indien de inham totaal is, dat wil zeggen, wanneer de momentele zonebreedte nul is, dan is een grens bereikt. De laatste positie en richting worden van de stack gehaald (5A1FH) en het opvullen begint weer op dat punt. Anders worden de momentele positie en richting op de stack gezet (5AC2H) en het opvullen begint weer links onderaan de inham.

Een linkse inham wordt automatisch verwerkt tijdens het zoeken naar de rechter grenslijn, en vergt dus geen speciale actie vanwege de opvulroutine.

adres 5AC2H

Deze routine wordt gebruikt door de routine die het "PAINT"-statement verwerkt, om de laatst beschreven schermpositie en de werkrichting op de stack van de Z80 te zetten. Het parameterblok, zes bytes groot , bestaat uit het volgende :

2 bytes ... momentele inhoud van CLOC
1 byte ... richting
1 byte ... inhoud van CMASK
2 bytes ... breedte van de zone

Na deze bewerking wordt gecontroleerd of er nog voldoende ruimte op de stack is (625EH).

adres 5ADCH

Deze routine wordt gebruikt door de routine die het "PAINT"-statement verwerkt, om uit te zoeken waar de rechter grenslijn ligt. De zonebreedte van de vorige lijn wordt in registerpaar DE aan de standaardroutine SCANR doorgegeven, waardoor bepaald wordt hoeveel pixels in de grenskleur oorspronkelijk overgeslagen mogen worden. Wat daarvan nog rest, wordt in SKPCNT gezet en het aantal pixels dat niet in die kleur stond, wordt in MOVCNT gezet.

adres 5AEDH

Deze routine wordt gebruikt door de routine die het "PAINT"-statement verwerkt, om de linker grenslijn op te zoeken. Het punt waar de zoektocht naar de rechter grenslijn stopte, wordt tijdelijk opgeslagen. Het startpunt wordt uit CSAVEA en CSAVEM gehaald, en tot courante schermpositie gemaakt. Dan wordt de linker grenslijn gezocht via de standaardroutine SCANL, die tegelijk de hele zone opvult. Het rechterpunt wordt teruggehaald en in CSAVEA en CSAVEM gezet.

adres 5B0BH

Deze routine wordt gebruikt door de routine die het "CIRCLE"-statement verwerkt, om het teken van de inhoud van registerpaar DE om te keren.

adres 5B11H

Deze routine verwerkt het "CIRCLE"-statement. Eerst worden de coördinaten van het middelpunt geëvalueerd (579CH), daarna de straal (520FH). Deze laatste wordt vermenigvuldigd (325CH) met SIN(PI/4) en in CNPNTS gezet. De inktkleur wordt ingevuld (584DH), de beginhoek geëvalueerd (5D17H) en in CSTCNT gezet. Daarna wordt de eindhoek geëvalueerd (5D17H) en in CENCNT gezet. Is de eindhoek kleiner dan de beginhoek, dan worden de twee waarden verwisseld en CPLOTF wordt niet-nul gemaakt. De asverhouding wordt geëvalueerd (4C64H) en indien die groter is dan 1 wordt de reciproke berekend (3267H) en wordt CSCLXY niet-nul gemaakt om aan te geven dat er een afplatting langs de X-as moet gebeuren. De asverhouding wordt met 256 vermenigvuldigd, in een geheel getal omgezet (2F8AH) en in ASPECT gezet als een binaire fractie van 1 byte. Registerparen DE en HL worden geladen met de startpositie op de omtrek van de cirkel (X = straal, Y = 0) en de controle gaat over in de hoofdlus.

adres 5BBDH

Dit is de hoofdlus van de cirkel-routine. Omdat een cirkel een hoge graad van symmetrie bezit, moeten enkel de coördinaten van de boog van nul tot vijfenveertig graden worden berekend. De overige zeven segmenten worden gemaakt door die punten te roteren en de spiegelen. De parametervergelijking voor een eenheids-cirkel, waarbij T de hoek is van nul tot PI/4, is :

```
X = COS(T)
Y = SIN(T)
```

Directe berekening met behulp van deze vergelijking, of met de overeenkomstige functie X = SQR(1-Y^2), is te traag. Daarom wordt de eerste afgeleide gebruikt :

```
dx/dy = -Y/X
```

Indien het startpunt bekend is (X = straal, Y = 0) kan de wijziging in het X-coördinaat voor elke wijziging met een eenheid van het Y-coördinaat berekend worden. Bovendien is het zo dat, omwille van de beperktheid van de grafische resolutie tot één pixel, we enkel moeten weten wanneer de som van de wijzigingen van het X-coördinaat 1 wordt, om vervolgens het X-coördinaat met 1 te verlagen. Vandaar geldt :

```
verlaag X indien (Y1/X)+(Y2/X)+(Y3/X)+... => 1
dus verlaag X indien      (Y1+Y2+Y3+...)/X => 1
dus verlaag X indien            Y1+Y2+Y3+... => X
```

Alles wat nodig is om te weten wanneer X moet veranderen, is de waarden van de Y-coördinaten op te tellen bij elke stap, tot de waarde van het X-coördinaat overschreden wordt. De cirkelroutine bewaart het X-coördinaat in registerpaar HL, het Y-coördinaat in registerpaar DE en de totaalsom tot dan toe in CRCSUM. Een equivalent BASIC programma om een cirkel te tekenen met een willekeurig gekozen straal van 160 pixels, is dit :

```
10 SCREEN 2
20 X=160:Y=0:CRCSUM=0
30 PSET (X,191-Y)
40 CRCSUM=CRCSUM+Y:Y=Y+1
50 IF CRCSUM<X THEN 30
60 CRCSUM=CRCSUM-X:X=X-1
70 IF X>Y THEN 30
80 CIRCLE (0,191),155
90 GOTO 90
```

De coördinatenparen die de hoofdlus produceert, zijn die van een "virtuele" cirkel. De spiegelingen rond de assen, de elliptische afplatting en de middelpunts-bepaling worden op een lager niveau verwerkt (5C06H).

adres 5C06H

Deze routine wordt door hoofdlus van de cirkelroutine gebruikt om een coördinatenpaar, in registerparen HL en DE, om te rekenen in acht symmetrisch liggende punten op het scherm. Om te

beginnen wordt het Y-coördinaat negatief gemaakt (5B0BH), waardoor hij in de X-as wordt gespiegeld, en de eerste vier punten worden berekend door opeenvolgende draaiingen over negentig graden in wijzerzin (5C48H). Dan wordt het Y-coördinaat opnieuw van teken veranderd (5B0BH) en er worden nog vier punten berekend (5C48H). Rotatie in wijzerzin wordt uitgevoerd door de X- en Y-coördinaten te verwisselen en het nieuwe Y-coördinaat negatief te maken. Het punt (40,10) wordt zo (10,-40). Indien de afplattingsfactor 0,5 is, bijvoorbeeld, dan zijn de acht punten de volgende :

1) X,-Y*0,5
2) -Y,-X*0,5
3) -X,Y*0,5
4) Y,X*0,5
5) Y,-X*0,5
6) -X,-Y*0,5
7) -Y,X*0,5
8) X,Y*0,5

Uit het bovenstaande kan geleerd worden dat, indien we de tekens van de coördinaten voor het ogenblik niet in acht nemen, er slechts vier termen meespelen. Daarom worden de termen X*0,5 en Y*0,5 vooraf klaargemaakt en de volledige reeks wordt berekend door de vier termen onderling te verwisselen en negatief te maken. Dit werkt sneller dan voor elk punt de afplattingsfactor te vermenigvuldigen met een van de termen (5CEBH). Voor een afplattingsfactor van 0,5 worden de beginwaarden als volgt opgezet : registerpaar HL = X; registerpaar DE = -Y*0,5; CXOFF = Y en CYOFF = X*0,5. De opeenvolgende punten worden berekend door de bewerkingen :

1) verwissel HL en CXOFF, maak HL negatief
2) verwissel DE en CYOFF, maak DE negatief

Parallel met de berekening van elk coördinaat van de cirkel, wordt het aantal punten dat nodig is om de start van het segment te bereiken waarin dat coördinaat ligt, bijgehouden in CPCNT8. Bij de start is dat 0, en bij elke rotatie van negentig graden wordt het verhoogd met 2*straal*SIN(PI/4). Bij het berekenen van elk van de acht punten wordt de Y-coördinaat ervan bij de inhoud van CPCNT8 opgeteld, en vergeleken met de beginhoek en de eindhoek om de bepalen wat er moet gebeuren. Ligt het punt tussen de twee hoeken, en is CPLOTF nul, of ligt het buiten de hoeken en is CPLOTF verschillend van nul, dan worden de coördinaten opgeteld bij die van het middelpunt van de cirkel (5CDCH) en het punt wordt geset via SCALXY, MAPXYC en SETC. Ligt het punt op een van de twee hoeken, en het overeenkomstige bit in CLINEF is geset, dan worden de coördinaten opgeteld bij die van het middelpunt van de cirkel (5CDCH) en er wordt een lijn getrokken naar dat middelpunt (593CH). Wordt aan geen van de genoemde voorwaarden voldaan, dan wordt gewoon verdergegaan naar het volgende punt.

adres 5CEBH

Deze routine vermenigvuldigt de coördinaatwaarde in registerpaar DE met de afplattingsfactor in ASPECT. Het resultaat wordt in registerpaar DE gezet. De standaardmethode van binair schuiven en optellen wordt gebruikt, maar om overflowproblemen te vermijden, wordt de bewerking uitgevoerd als twee vermenigvuldigingen van één byte.

adres   5D17H

Deze routine wordt door de "CIRCLE"-routine gebruikt om een operand die een hoek aangeeft, om te zetten in de vorm die de hoofdlus van de cirkelroutine kan gebruiken. Het resultaat wordt in registerpaar DE gezet. Hoewel de gebruikte methode in principe goed is, en één driehoeksmeetkundige berekening per hoek vermijdt, zijn de geproduceerde resultaten niet nauwkeurig. Dit wordt aangetoond door het volgende voorbeeld, waarin een lijn getrokken wordt naar het echte dertig graden-punt op een cirkel - omtrek :

```
10 SCREEN 2
20 PI=4*ATN(1)
30 CIRCLE(100,100),80,,PI/6
40 LINE(100,100)-(100+80*COS(PI/6),100-80*SIN(PI/6))
50 GOTO 50
```

De routine hoort eigenlijk het aantal punten te berekenen dat door de hoofdlus van de cirkelroutine hoort te worden getekend voordat de gevraagde hoek bereikt wordt. Dit kan berekend worden door eerst te bedenken dat er INT(HOEK/(PI/4)) segmenten van vijfenveertig graden komen voor het segment dat de gevraagde hoek bevat. Daarenboven zal elk segment van vijfenveertig graden STRAAL*SIN(PI/4) punten bevatten, aangezien dat de waarde is van de laatste Y-coördinaat. Daaruit volgt dat het aantal punten, nodig om het beginpunt te bereiken van het segment dat de hoek bevat, het product van deze twee getallen is. Het totale aantal is bekend, als dit berekend aantal opgeteld wordt bij het aantal punten, dit is nodig om een eventuele resthoek binnen het laatste segment af te werken. Dat aantal wordt berekend als : STRAAL*SIN(resthoek) punten.

Jammer genoeg berekent de routine het aantal punten binnen een segment door lineaire benadering, vanaf de totale grootte van het segment. Daarbij wordt uitgegaan van de foutieve veronderstelling dat opeenvolgende punten gelijke hoeken onderspannen. Vandaar dat in het hoger gegeven voorbeeld, het aantal punten dat berekend wordt voor de hoek, door de formule 30/45*(80*0,707107) 37 oplevert in plaats van het correcte aantal van 40. De fout die door de routine gemaakt wordt is dus maximaal in het midden van elk segment van vijfenveertig graden, en wordt nul bij de eindpunten.

adres   5D6EH

Deze routine verwerkt het "DRAW"-statement. Registerpaar DE wordt geladen met het startadres van de tabel met commando's op 5D83H, en de controle wordt overgedragen aan de macro-ontleder (566CH).

adres   5D83H

Deze tabel bevat de toegelaten commando-letters met de overeenkomstige adressen, van het "DRAW"-statement. De commando's die een parameter vergen, en derhalve bit 7 geset hebben in de tabel, worden met een asterisk aangeduid.

COMMANDO	ADRES
U*	5DB1H
D*	5DB4H
L*	5DB9H
R*	5DBCH
M	5DD8H
E*	5DCAH
F*	5DC6H
G*	5DD1H
H*	5DC3H
A*	5E4EH
B	5E46H
N	5E42H
X	5782H
C*	5E87H
S*	5E59H

adres 5DB1H

Deze routine verwerkt het "U"-commando voor het "DRAW"-statement. De werking van de overige commando's ("D","L","R","E","F","G" en "H") is erg gelijklopend. Daarom geven we geen afzonderlijke beschrijving van de routines.

De facultatieve numerieke parameter wordt door de macro-ontleder in register DE aangebracht. Deze oorspronkelijke parameter wordt door één van de routines omgerekend in een horizontale verplaatsing, die in registerpaar BC wordt gezet, en in een vertikale verplaatsing, die in registerpaar DE wordt gezet. Indien bijvoorbeeld een beweging naar links of naar boven op het scherm moet uitgevoerd worden, wordt de parameter negatief gemaakt (5B0BH). Moet een diagonale beweging worden uitgevoerd, dan wordt de parameter gedupliceerd, zodat gelijke horizontale en vertikale bewegingen het resultaat zijn. Wanneer de parameters voorbereid zijn, gaat de controle over op de "LINE"-routine (5DFFH).

adres 5DD8H

Deze routine verwerkt het "M"-comando voor het "DRAW"-statement. Het karakter na de commandoletter wordt onderzocht, en dan worden de twee parameters uit de commandostring gelezen (5719H). Is het eerste karakter een "+" of een "-", dan worden de parameters beschouwd als afstanden, en ze worden op schaal gebracht (5E66H), vervolgens herhaaldelijk over negentig graden geroteerd, zoals in DRWANG bepaald, en tenslotte bij de courante grafische coördinaten opgeteld (5CDCH) om het eindpunt te berekenen. Indien DRWFLG aangeeft dat de "B"-mode niet actief is, wordt een lijn getrokken (5CCDH) vanaf de courante grafische coördinaten tot het eindpunt. Geeft DRWFLG aan dat de "N"-mode niet actief is, dan worden de eind-coördinaten in GRPACX en GRPAXY gezet, waardoor zij de courante grafische coördinaten worden. Tenslotte wordt DRWFLG op nul gezet, waardoor de "B" en "N" modes buiten werking gesteld worden, en de routine stopt.

adres 5E42H

Deze routine verwerkt het "N"-commando voor het "DRAW"-statement. DRWFLG wordt eenvoudig op 40H gezet. (N="Negative" : in plaats van begin naar eindpunt, wordt de lijn van eindpunt naar beginpunt getrokken; hierbij worden de begincoördinaten bewaard)

adres 5E46H

Deze routine verwerkt het "B"-commando voor het "DRAW"-statement. DRWFLG wordt eenvoudig op 80H gezet.(B="Blank" : er wordt geen zichtbare lijn getrokken, maar de coördinaten worden wel verplaatst)

adres 5E4EH

Deze routine verwerkt het "A"-commando voor het "DRAW"-statement. De grootte van de parameter wordt gecontroleerd en in DRWANG gezet.(A="Angle" : hiermee wordt de hoek aangegeven, waaronder getekend moet worden)

adres 5E59H

Deze routine verwerkt het "S"-commando voor het "DRAW"-statement. De grootte van de parameter wordt gecontroleerd en in DRWSCL gezet.(S="Scale" : hiermee kan men een schaal opgeven waarin getekend moet worden)

adres 5E66H

Deze routine wordt gebruikt door de "U", "D", "L", "R", "E", "F", "G", "H", en "M"-commando's (dit laatste enkel in offset-mode) van het "DRAW"-statement, om de afstanden in registerpaar DE op schaal te brengen met behulp van de inhoud van DRWSCL. Indien DRWSCL nul bevat, stopt de routine. In het andere geval wordt de afstand vermenigvuldigd door herhaalde optellingen en daarna door vier gedeeld (59B4H). Om het op schaal brengen te vermijden, moet een "S0" of "S4"-commando worden gebruikt.

adres 5E87H

Deze routine verwerkt het "C"-commando voor het "DRAW"-statement. De parameter wordt in ATRBYT gezet via de standaardroutine SETATR. Het MSB van de parameter wordt niet gecontroleerd, zodat illegale waarden zoals "C265" zonder foutmelding worden geaccepteerd.(C="Color" : tekenen in een bepaalde kleur).

adres 5E91H

Deze routine wordt door de "PAINT"-routine gebruikt om, via de standaardroutine SCALXY, te controleren of de coördinaten in registerparen BC en DE binen het scherm vallen. Indien dit niet zo is, wordt de foutmelding "Illegal function call" gegeven (475AH).

adres SE9FH

Deze routine verwerkt het "DIM"-statement. Er wordt een terugkeer naar 5E9AH voorzien, zodat meervoudige arrays verwerkt kunnen worden. DIMFLG wordt niet-nul gemaakt, en de controle gaat over in de routine die variabelen opzoekt.

adres 5EA4H

Dit is de routine die een variabele opzoekt in het geheugen. Bij de start bevat registerpaar HL het adres van het eerste karakter van de naam van de variabele in het programmageheugen. Op het einde van de routine bevat registerpaar HL het adres van het karakter nã de naam, en registerpaar DE het adres van het eerste byte van de inhoud van die variabele in het variabelengeheugen. Het eerste karakter van de naam wordt gelezen. Daarvan wordt gecontroleerd of het een hoofdletter is (64A7H) en dan wordt het in register C gezet. Het facultatieve tweede karakter wordt in register B gezet. Dit mag een cijfer of een letter zijn; is er geen tweede karakter, dan wordt een standaardwaarde nul aangenomen. Staan er nog meer karakters, dan worden die overgeslagen. Volgt een type-aanduiding na de naam van de variabele ("%", "$", "!" of "#") dan wordt die omgezet in de overeenkomstige typecode (2, 3, 4, 8) en in VALTYP gezet. Zoniet wordt het standaardtype van die variabele uit DEFTBL gelezen. Daarbij wordt de eerste letter van de naam gebruikt om het correcte adres te bepalen.

Nu wordt in SUBFLG gekeken hoe een eventueel volgend getal tussen haakjes behandeld moet worden. Normaal is deze vlag 0, maar ze wordt gewijzigd door de routines die de statements "ERASE" (01H), "FOR" (64H), "FN" (80H) en "DEF FN" (80H) verwerken, om een bepaalde werkwijze te forceren. Indien het "ERASE"-statement voorkomt, wordt de controle rechtstreeks aan de array-zoekroutine overgedragen (5FE8H) : dan moet geen getal tussen haakjes aanwezig zijn. Indien het om de statements "FOR", "FN" of "DEF FN" gaat, wordt rechtstreeks naar de zoekroutine voor gewone variabelen gesprongen (5F08H), zonder te controleren of er een getal tussen haakjes volgt. Is het een normale situatie, wordt in de programmatekst gekeken of er een "(" of een "[" staat. Is dat zo, dan wordt de controle overgedragen aan de array-zoekroutine (5FBAH); zo niet, loopt de controle door in de zoekroutine voor gewone variabelen.

adres 5F08H

Deze routine zoekt een gewone variabele op in het variabelengeheugen. Er bestaan vier types van gewone variabelen, die elk bestaan uit een "header", gevolgd door de inhoud van de variabele. In het eerste byte van de header staat de typecode, in de volgende twee bytes de naam van de variabele. De inhoud van de variabele zal bij een numerieke variabele bestaan uit één van de drie standaardvormen, en bij een stringvariabele uit de lengte (1 byte) en het adres van de string (2 bytes). Hieronder volgt een voorbeeld van elk van de vier types :

02H	"A"	"B"	LSB	MSB

geheel getal

03H	"A"	"B"	LEN	LSB	MSB

string

04H	"A"	"B"	EE	CC	CC	CC

enkele precisie

08H	"A"	"B"	EE	CC	CC	CC	CC	CC	CC	CC

dubbele precisie

Fig. 46 : gewone variabelen

Eerst wordt in NOFUNS gecontroleerd of momenteel een zelf-gedefinieerde functie geëvalueerd wordt. Is dat zo, dan wordt eerst in PARM1 naar de bewuste variabele gezocht. Wordt die daar niet gevonden, dan gaat de zoektocht verder in het variabelengeheugen. Er wordt lineair gezocht : de twee karakters van de naam plus de typecode van elke variabele in één van die twee geheugengebieden worden vergeleken met de referentie-naam en het type, tot ofwel een identiek signalement wordt gevonden, ofwel het einde van het variabelengeheugen werd bereikt. Indien een identieke variabele werd gevonden, stopt de routine met in registerpaar DE het adres van het eerste byte van de gevonden variabele. Werd de variabele niet gevonden, dan wordt het array-geheugen hoger in het geheugen gezet, en de nieuwe variabele wordt bij het variabelengeheugen gevoegd, en op nul geïnitialiseerd.

Er zijn twee gevallen waarin géén nieuwe variabele gecreëerd wordt. Indien de variabele wordt gezocht door de "VARPTR"-functie, en dat wordt bepaald door het berekende adres te onderzoeken, dan wordt geen nieuwe variabele aangemaakt, indien hij niet werd gevonden in het geheugen. In plaats ervan wordt registerpaar DE op nul gezet (5F61H), wat de foutmelding "Illegal function call" oplevert. De tweede uitzondering is, wanneer de variabele wordt gezocht door de Factor Evaluator, wanneer een nieuwe variabele in een uitdrukking wordt ingevoerd. In dat geval wordt DAC op nul gesteld, voor een numerieke variabele, of geladen met het adres van een signalement dat nul als lengte bevat, indien het om een string gaat. Dat levert nul op als resultaat van de zoektocht (5FA7H). Deze manier van werken voorkomt dat de Expression Evaluator een nieuwe variabele aanmaakt. "VARPTR" is de enige functie die rechtstreeks een variabele als argument neemt, in plaats van dat via een uitdrukking te doen; daarom moet ze apart beschermd worden. Zonder die ingebouwde bescherming zou een toewijzing van een array-variabele door het "LET"-statement mislukken, omdat het adres van dat array gewijzigd zou worden door het aanmaken van een gewone variabele tijdens de evaluatie van een uitdrukking.

adres 5FBAH

Deze routine zoekt een variabele in een array op. Er bestaan vier types van arrays, die elk bestaan uit een header plus een aantal elementen. Het eerste byte in de header bevat de typecode. De volgende twee bevatten de naam van het array en de volgende twee de afstand tot aan het begin van het volgende array. Dan volgt één byte waarin het aantal dimensies van de array staan. Daarna komt de lijst van het aantal elementen. Per twee bytes wordt aangegeven hoeveel elementen er maximaal per dimensie kunnen voorkomen. Die hoeveelheden worden opgeslagen in omgekeerde volgorde : de eerste twee bytes komen overeen met de laatst opgegeven dimensie. De inhoud van elk element van een array is dezelfde als die van de overeenkomstige gewone variabele. Hieronder wordt het array AB%(3,4) getoond, waarin elk element geïdentificeerd wordt door de twee indexen. De hoge geheugenadressen liggen bovenaan de bladzijde.

```
(0,4)  (1,4)  (2,4)  (3,4)
(0,3)  (1,3)  (2,3)  (3,3)
(0,2)  (1,2)  (2,2)  (3,2)
(0,1)  (1,1)  (2,1)  (3,1)
(0,0)  (1,0)  (2,0)  (3,0)
```

			LENGTE		DIM	TELLER		TELLER	
02H	"A"	"B"	2DH	00H	02H	05H	00H	04H	00H

Fig. 47 : array van gehele getallen

Elke index wordt geëvalueerd, in een geheel getal omgezet (4755H) en op de stack van de Z80 gezet, tot een ")" of een "]" wordt gevonden. Het sluit-haakje mag verschillen van het open-haakje. Daarna wordt lineair gezocht in het array-geheugen naar de opgegeven naam en het type. Staat die al in het geheugen, dan wordt in DIMFLG gekeken of een "DIM"-statement wordt uitgevoerd. Is dat zo, dan wordt de foutmelding "Redimensioned array" gegeven (405EH). Indien een "ERASE"-statement wordt uitgevoerd, stopt de routine met in registerpaar BC het startadres van het array (3279H). In het andere geval wordt het aantal dimensies van het array vergeleken met het aantal indexen en indien deze twee verschillen, wordt de foutmelding "Subscript out of range" gegeven. Na alle controles wordt de controle overgedragen naar het punt waar het adres van een element in het array wordt berekend (607DH).

Werd de opgegeven variabele niet gevonden, tijdens de uitvoering van een "ERASE"-statement, dan wordt de foutmelding "Illegal function call" gegeven (475AH). Ging het niet om een "ERASE"-statement, dan wordt een nieuw array toegevoegd op het einde van het array-geheugen. Dan begint de initialisatie van het array. Eerst wordt de naam (twee karakters) opgeslagen, daarna de typecode en de dimensionering (het aantal dimensies), gevolgd door het aantal elementen per dimensie. Geeft DIMFLG aan dat een "DIM"-statement wordt uitgevoerd, dan wordt het aantal elementen bepaald door de indexen. Wordt het array automatisch gecreëerd, bijvoorbeeld door een statement zoals "A(1,2,3)=5", dan wordt een standaardwaarde van elk 10 aangehouden. Tijdens de opslag

van elke index wordt de totale omvang van het array bijgehouden in registerpaar DE, door opeenvolgende vermenigvuldigingen (314AH) van de index met het aantal bytes per element (het type van het array). Nu wordt gecontroleerd of nog zoveel bytes beschikbaar zijn in het geheugen (6267H). Zo ja, wordt STREND verhoogd, het nieuwe arraygebied wordt met nullen gevuld en de omvang van het array wordt, in licht gewijzigde vorm, opgeslagen na de twee karakters van de naam. Tenzij het array automatisch werd aangemaakt, wat inhoudt dat het adres van het element berekend moet worden, stopt de routine.

Op dit punt wordt het adres van een element berekend. Een bepaald element in een array wordt gelocaliseerd door vermenigvuldiging (314AH) van indexen, aantal elementen en grootte per element. Dit kan op een aantal verschillende manieren gebeuren. Daarom wordt de manier die door deze routine gebruikt wordt, het beste geïllustreerd aan de hand van een voorbeeld. Het element (1,2,3) in een array (4,5,6) wordt eerst gelocaliseerd door de uitdrukking (((3*5)+2)*4)+1. Het resultaat daarvan wordt dan vermenigvuldigd met het aantal bytes per element (type) en dit product opgeteld bij het startadres van het array. Dit geeft dan het adres van het bedoelde element. De berekening gebeurt op een geoptimaliseerde manier, die het aantal stappen tot een mimimum herleidt. Het equivalent is de evaluatie van (3*(4*5))+(2*4)+(1). Het adres van het element staat op het einde van de routine in registerpaar DE.

adres 60B1H

Deze routine verwerkt het "PRINT USING"-statement. De controle wordt naar dit punt overgegeven door de algemene "PRINT"-routine nadat er voor gezorgd werd dat de output naar het juiste randapparaat werd geleid. Op het einde van de routine wordt de controle weer overgedragen naar het eindpunt van de algemene "PRINT"-routine (4AFFH), om de normale video-output te herstellen.

Eerst wordt de opgegeven string geëvalueerd (4C65H), en het adres en de lengte van de string uit het signalement gehaald. Het adres waar de Interpreter aan het werk was, wordt tijdelijk opgeslagen. Elk karakter van de string wordt bekeken, tot één van de karakters die in de print-mal kunnen staan, wordt gevonden. Hoort een bepaald karakter niet in een mal thuis, dan wordt het gewoon via de standaardroutine OUTDO afgedrukt. Vanaf de start van een mal wordt elk karakter bekeken, tot opnieuw een karakter wordt ontdekt dat er niet in thuishoort. Daar wordt de controle overgedragen aan de routine die de numerieke output verzorgt (6192H) of de routine die strings afdrukt (6211H).

In beide gevallen wordt het adres van de Interpreter hersteld in registerpaar HL, en daarna wordt de volgende operand geëvaluerd (4C64H). Voor het afdrukken van een getal, wordt de informatie die werd verkregen uit het onderzoek van de mal, in registers A, B en C doorgegeven aan de numerieke omzettings-routine (3426H), en de daaruit geproduceerde string wordt afgedrukt (6678H). Voor het afdrukken van een string wordt het vereiste aantal karakters in register C doorgegeven aan de "LEFT$"-routine (6868H) en de aldus verkregen string afgedrukt (667BH). Bij beide types wordt dan nog gecontroleerd in de programmatekst en in de "USING"-string of er nog meer karakters volgen. Worden geen

operands meer gevonden, dan stopt de routine. Werd de "USING"-string tot het einde toe afgewerkt, en volgen nog meer operands, dan begint de routine weer op adres 60BFH, zoniet gaat de aftasting verder vanaf het laatste bereikte adres, voor de volgende operand (60F6H).

adres 6250H

Deze routine wordt door de Hoofdlus van de Interpreter en de variabelen-zoekroutine gebruikt om een geheugenblok hoger in het geheugen te zetten. Eerst wordt gecontroleerd of er nog voldoende geheugen vrij is (6267H), en dan wordt het blok verzet. Registerpaar BC bevat het hoogste adres van het te verzetten blok, registerpaar HL het hoogste adres van het verzette blok. De bewerking stopt wanneer de inhoud van registerpaar BC gelijk is aan die van registerpaar DE.

adres 625EH

Deze routine controleert of er voldoende ruimte vrij is tussen het laatste byte van het array-geheugen en het laagste byte van de stack van de Z80. Bij de start van de routine bevat register C het aantal woorden dat de gebruiker van deze routine nodig heeft. Wordt daardoor de vrije ruimte kleiner dan tweehonderd bytes, dan wordt de foutmelding "Out of memory" gegeven.

adres 6286H

Deze routine verwerkt het "NEW"-statement. TRCFLG, AUTFLG en PTRFLG worden op nul gezet en het schakeladres 0000H wordt aan het begin van het programmageheugen gezet. In VARTAB wordt het adres gezet van het byte dat volgt op dat schakeladres, en de controle over over in de "RUN"-"CLEAR"-routine.

adres 629AH

Deze routine wordt door de "NEW"-, "RUN"- en "CLEAR"-routines gebruikt, om de systeemvariabelen te initialiseren. Alle interrupt-functies worden op nul gezet (636EH) en in DEFTBL worden de standaard-types voor de variabelen op dubbele precisie gezet. RNDX wordt op de startwaarde gezet (2C24H) en ONEFLG, ONELIN en OLDTXT worden nul. MEMSIZ wordt in FRETOP gecopieerd, om het stringgeheugen leeg te maken, en DATPTR wordt geladen met het startadres van het programmageheugen (63C9H). De inhoud van VARTAB wordt in ARYTAB en STREND gecopieerd, om alle variabelen te wissen. Alle In/Out-buffers worden gesloten (6C1CH) en NLONLY wordt op de startwaarde gezet. Met behulp van STKTOP worden SAVSTK en de Stack Pointer van de Z80 op hun beginwaarden gezet. In TEMPPT wordt het startadres gezet van TEMPST, om alle string-signalementen te wissen. De printer wordt afgesloten (7304H) en de output wordt naar het scherm geleid (4AFFH). Tenslotte worden PRMLEN, NOFUNS, PRMLN2, FUNACT, PRMSTK en SUBFLG nul gemaakt, en de routine stopt.

adres 631BH

Deze routine wordt gebruikt door de "DEVICE ON"-statements, om een interrupt-bron in werking te stellen. In registerpaar HL staat het adres van het statusbyte in TRPTBL dat bij het opgegeven apparaat hoort. De interrupts worden in werking gesteld door bit Ø van het statusbyte te setten. Daarna worden bits 1 en 2 bekeken. Indien het opgegeven apparaat gestopt werd, en een interrupt ontvangen werd, wordt ONGSBF opgehoogd (634FH), zodat de Verwerkingslus de interrupt zal verwerken, op het einde van het statement. Tenslotte wordt bit 1 van het statusbyte gereset, om een eventuele stop-status op te heffen.

adres 632BH

Deze routine wordt door de "DEVICE OFF"-statements gebruikt, om een interrupt-bron buiten werking te stellen. Het adres van het statusbyte in TRPTBL dat bij het opgegeven apparaat hoort, staat in registerpaar HL. In bits Ø en 2 van het statusbyte wordt gecontroleerd of er sinds het einde van het laatste statement een interrupt is opgetreden. Zo ja, wordt ONGSBF met één verlaagd (6362H) om te verhinderen dan de Verwerkingslus hem zou uitvoeren. Tenslotte wordt bit 1 van het statusbyte geset.

adres 6331H

Deze routine wordt gebruikt door de "DEVICE STOP"-statements, om de verwerking van interrupts, afkomstig van een interrupt-bron, op te schorten. Het adres van het statusbyte in TRPTBL dat bij het opgegeven apparaat hoort, staat in registerpaar HL. In bits Ø en 2 wordt gecontroleerd of er sinds het einde van het laatste statement een interrupt werd ontvangen. Zo ja, wordt ONGSBF met één verlaagd (6362H) om te verhinderen dat de Verwerkingslus hem zou uitvoeren. Tenslotte wordt bit 1 van het statusbyte geset.

adres 633EH

Deze routine wordt gebruikt door de "RETURN"-routine, om de tijdelijke stop-status op te heffen, die tijdens BASIC interupt-subroutines wordt ingesteld. Het adres van het statusbyte in TRPTBL dat bij het opgegeven apparaat hoort, staat in registerpaar HL. In bits Ø,1 en 2 wordt gecontroleerd of sinds het aanroepen van de subroutine een interrupt was opgetreden, die gestopt werd. Is dat zo, dan wordt ONGSBF opgehoogd (634FH) zodat de Verwerkingslus de interrupt op het einde van het statement zal verwerken. Daarna wordt bit 1 gereset. Dat houdt in dat een "DEVICE STOP"-statement in een dergelijke subroutine, geen effect zal hebben.

adres 6358H

Deze routine wordt gebruikt door de interrupt-verwerker van de Verwerkingslus (6389H), om een interrupt die optreedt voordat de BASIC-subroutine wordt verwerkt, ongedaan te maken. Het adres van het statusbyte in TRPTBL dat bij het opgegeven apparaat hoort, staat in registerpaar HL. ONGSBF wordt met één verlaagd en bit 2 van het statusbyte wordt gereset.

adres 636EH

Deze routine wordt door de "RUN"-"CLEAR"-routines (629AH) gebruikt om alle interrupts buiten werking te stellen. De achtenzeventig bytes van TRPTBL en de tien bytes van FNKFLG worden nul.

adres 6389H

Dit is de interrupt-verwerker van de Verwerkingslus. Eerst wordt in ONEFLG gecontroleerd of er op dat ogenblik een fout verwerkt wordt. Zo ja, stopt de routine : er worden geen interrupts verwerkt tot de fout opgeheven is. Zoniet, wordt CURLIN bekeken. Is de Interpreter een direct commando aan het verwerken, dan stopt de routine. Is alles in orde, dan worden de zesentwintig statusbytes in TRPTBL bekeken, om de eerste actieve interrupt te vinden. Dit houdt in dat apparaten, die voor in de tabel staan, voorrang krijgen op de volgende. Wordt een actieve statusbyte gevonden, dat is er een met bits 0 en 2 geset, dan wordt het overeenkomstige adres uit TRPTBL gelezen en in registerpaar DE gezet. De interrupt wordt gewist (6358H) en het apparaat gestopt (6331H) waarna de controle overgedragen wordt aan de "GOSUB"-routine (47CFH).

adres 63C9H

Deze routine verwerkt het "RESTORE"-statement. Wordt dit niet gevolgd door een operand die een regelnummer inhoudt, dan wordt in DATPTR het startadres van het programmageheugen gezet. In het andere geval wordt de operand gelezen (4769H), en het programmageheugen doorzocht tot de opgegeven regel wordt gevonden (4295H). Het startadres van die regel wordt dan in DATPTR gezet.

adres 63E3H

Deze routine verwerkt het "STOP"-statement. Volgt daarna nog tekst, dan wordt de controle overgedragen aan de "STOP ON/OFF/STOP"-routine (77A5H). Zoniet, wordt register A met 01H geladen en de controle loopt over in de "END"-routine.

adres 63EAH

Deze routine verwerkt het "END"-statement. Ze wordt ook, met een verschillend toegangsadres, gebruikt door het "STOP"-statement, bij een "CTRL/STOP" en bij het beëindigen van een programma. Eerst wordt ONEFLG nul gemaakt en dan - alleen bij "END" - worden alle In/Out-buffers gesloten (6C1CH). De huidige positie in de programmatekst wordt in SAVTXT en OLDTXT gezet, en het courante regelnummer in OLDLIN, waar een eventueel volgend "CONT"-statement ze kan ophalen. De printer wordt stilgelegd (7304H). Naar het scherm wordt een "CR"/"LF"-combinatie gestuurd (7323H) en in registerpaar HL wordt het adres geladen van de tekst "Break", op adres 3FDCH. Gaat het om een "END"-statement of werd het einde van de programmatekst bereikt, dan wordt de controle overgegeven aan het "Ok"-punt van de Hoofdlus (411EH). Werd een "CTRL/STOP" verwerkt, dan gaat de controle over naar het einde van de foutverwerker (40FDH), om de tekst "Break" af te drukken.

adres 6424H

Deze routine verwerkt het "CONT"-statement. Tenzij de inhoud van OLDTXT nul is, wat de foutmelding "Can't CONTINUE" tot gevolg heeft, wordt die in registerpaar HL geladen, en de inhoud van OLDLIN in CURLIN. Daarna wordt de controle overgedragen aan de Verwerkingslus, die vanaf het opgegeven adres verder gaat met de verwerking van het programma. De verwerking van een programma kan niet worden voortgezet indien "CTRL/STOP" werd ingedrukt tijdens de verwerking van een statement, dus via de standaard-routine CKCNTC, maar wel dat dat tussen twee statements in gebeurde.

adres 6438H

Deze routine verwerkt het "TRON"-statement. In TRCFLG wordt een van nul verschillende waarde gezet.

adres 6439H

Deze routine verwerkt het "TROFF"-statement. In TRCFLG wordt nul gezet.

adres 643EH

Deze routine verwerkt het "SWAP"-statement. De eerste variabele wordt opgezocht (5EA4H) en de inhoud ervan in SWPTMP gecopieerd. Het adres van die variabele en dat van het einde van het variabelengeheugen worden tijdelijk opgeslagen. Nu wordt de tweede variabele gezocht (5EA4H) en het type ervan vergeleken met dat van de eerste variabele. Zijn ze verschillend, dan wordt de foutmelding "Type mismatch" gegeven (406DH). Het huidige eindadres van het variabelengeheugen wordt nu vergeleken met het vroegere adres. Verschillen ze, dan wordt de foutmelding "Illegal function call" geproduceerd (475AH). Tenslotte wordt de inhoud van de tweede variabele gecopieerd naar het adres van de eerste (2EF3H) en daarna de inhoud van SWPTMP naar het adres van de tweede (2EF3H).

De uitgevoerde controles impliceren dat indien de tweede variabele een gewone variabele is, en niet in een array hoort, hij moet bestaan vooraleer het "SWAP"-statement wordt uitgevoerd, zoniet wordt een foutmelding gegeven. Daar is een reden voor. Stel dat de eerste variabele een element van een array was. Dan zou het toewijzen van een nieuwe gewone variabele het arraygeheugen verplaatsen, waardoor het berekende adres van de eerste variabele ongeldig wordt. Merk op dat de combinatie van een gewone eerste variabele en een nieuw toe te wijzen gewone variabele, eveneens geweigerd wordt, hoewel dit niet nodig zou zijn.

adres 6477H

Deze routine verwerkt het "ERASE"-statement. Eerst wordt SUBFLG op 01H gezet, ten behoeve van de variabelen-zoekroutine, en het array opgezocht (5EA4H). Alle arrays die volgen op het gezochte, worden naar beneden verhuisd, en STREND wordt op zijn nieuwe, lagere waarde gezet. Vervolgens wordt in de programmatekst gekeken of er een komma volgt; zo ja, dan keert de controle terug naar de start van deze routine.

adres 64A7H

Deze routine controleert of het karakter waarvan het adres in registerpaar HL staat, een hoofdletter is. Zo ja, wordt vlag NC gezet.

adres 64AFH

Deze routine verwerkt het "CLEAR"-statement. Volgt er geen operand, dan gaat de controle over op de "RUN-CLEAR"-routine (62A1H), om alle bestaande variabelen te wissen. In het andere geval wordt de operand die het stringgeheugen bepaalt, geëvalueerd (4756H), en dan de facultatieve operand die de geheugengrens aangeeft (542FH). Deze laatste waarde wordt gecontroleerd, en indien ze kleiner is dan 8000H of groter dan F380H wordt de foutmelding "Illegal function call" gegeven. De nodige ruimte voor de In/Out-buffers (267 bytes elk) en het stringgeheugen worden van de laatste operand afgetrokken, en indien er tot het startadres van het variabelengeheugen minder dan 160 bytes overblijven, wordt de foutmelding "Out of memory" gegeven (6275H). Is alles in orde, dan worden de nieuwe waarden van HIMEM, MEMSIZ en STKTOP ingevuld, en de overige pointers worden via de "RUN-CLEAR"-routine aangepast (62A1H). De opslagruimte voor de In/Out-buffers wordt opnieuw toegewezen (7E6BH) en de routine stopt.

Jammer genoeg worden, indien een nieuwe geheugengrens wordt opgegeven, MEMSIZ en STKTOP fout berekend. Daardoor wordt de bovengrens van het stringgeheugen 1 byte te hoog gezet. Dit wordt aangetoond door het volgende programma, waarin een niet-toegelaten string wordt geaccepteerd :

```
10 CLEAR 200,&HF380
20 A$=STRING$(201,"A")
30 PRINT FRE("")
```

Er zou op adres 64EBH een extra DEC HL-instuctie moeten staan. Het ontbreken daarvan maakt dat de nieuwe waarden van MEMSIZ en STKTOP oorspronkelijk één byte te hoog staan. Wanneer de "RUN-CLEAR"-routine wordt aangeroepen, wordt MEMSIZ in FRETOP gecopieerd, dat is het hoogste adres van het stringgeheugen. Daardoor is dit ook 1 byte te hoog. MEMSIZ en STKTOP worden correct herberekend tijdens het opnieuw setten van de bufferpointers, maar FRETOP wordt gelaten zoals het was. Wanneer nu het "FRE" - statement op regel 30 wordt uitgevoerd, en de "garbage collection" ingezet wordt, komt FRETOP op zijn juiste waarde maar doordat de string 1 byte langer is dan het stringgeheugen, wordt als vrije geheugenruimte "-1 byte" opgegeven. Om alle systeemvariabelen correct te setten, moet na elke wijziging in de bovenste geheugengrens, een "CLEAR"-statement zonder operands volgen.

adres 6520H

Deze routine berekent het verschil tussen de inhoud van registerparen HL en DE. Ze is identiek aan het stukje code tussen 64ECH en 64F1H, en wordt helemaal niet gebruikt.

adres 6527H

Deze routine verwerkt het"NEXT"-statement. Volgt er tekst op dit statement, dan wordt de lusvariabele gelocaliseerd (5EA4H), zoniet wordt een adres nul aangenomen. De stack van de Z80 wordt doorzocht naar het overeenkomstige "FOR"-parameterblok (3FE2H). Wordt dit niet gevonden, of wordt eerst een "GOSUB"-blok gevonden, dat wordt de foutmelding "NEXT without FOR" gegeven (405BH). Wordt het blok wél gevonden, dan wordt het gedeelte van de stack dat werd doorlopen, samen met andere "FOR"-blokken die er in voorkwamen, weggehaald. Het type van de lusvariabele wordt uit het parameterblok gelezen, waaruit de precisie bepaald wordt waarmee de volgende bewerkingen zullen plaatsvinden.

De STEP-waarde wordt uit het parameterblok gelezen en bij de huidige waarde van de lusvariabele geteld (3172H, 324EH of 2697H), die dan aangepast wordt. De nieuwe waarde wordt nu vergeleken (2F4DH, 2F21H of 2F5CH) met de eindwaarde die in het parameterblok staat, om te bepalen of de lus afgewerkt is (65B6H). Is de STEP-waarde positief, dan zal de lus eindigen indien de nieuwe luswaarde GROTER is dan de grenswaarde. Is de STEP-waarde negatief, dan eindigt de lus indien de nieuwe luswaarde KLEINER is dan de grenswaarde. Is de lus niet afgewerkt, dan wordt het programma-adres en regelnummer uit het parameterblok gelezen en de controle gaat over naar de Verwerkingslus (45FDH). Is de lus wél afgewerkt, dan wordt het parameterblok van de stack gewist en tenzij er nog meer tekst volgt die de controle terug naar het begin van de routine stuurt, gaat de controle over naar de Verwerkingslus om het volgende statement de verwerken (4601H).

adres 65C8H

Deze routine wordt door de Expression Evaluator gebruikt om de relatie (<>=) te vinden tussen twee string-operands. Het adres van het eerste signalement wordt op de stack van de Z80 verwacht, en het adres van het tweede in DAC. Het resultaat komt in register A en de vlaggen, zoals bij de numerieke routines :

string 1 = string 2 ... A = 00H, vlag Z, NC
string 1 < string 2 ... A = 01H, vlag NZ, NC
string 1 > string 2 ... A = FFH, vlag NZ, C

De vergelijking begint bij het eerste karakter van elke string, en loopt door tot de twee karakters verschillen of tot één van de strings doorlopen' is.Dan keert de controle terug op de Expression Evaluator (4F57H), om het resultaat als een "waar" of "onwaar"-waarde in DAC te zetten.

adres 65F5H

Deze routine wordt door de Factor Evaluator gebruikt om de "OCT$"-functie toe te passen op een operand in DAC. Eerst wordt het getal omgezet in tekstvorm, en in FBUFFR gezet (371EH), en dan wordt de overeenkomstige string gemaakt (6607H).

adres 65FAH

Deze routine wordt door de Factor Evaluator gebruikt om de "HEX$"-functie toe te passen op een operand in DAC. Eerst wordt

het getal in tekstvorm omgezet en in FBUFFR gezet (3722H), en dan wordt de overeenkomstige string gemaakt (6607H).

adres 65FFH

Deze routine wordt door de Factor Evaluator gebruikt om de "BIN$"-functie toe te passen op een operand in DAC. Eerst wordt het getal in tekstvorm omgezet en in FBUFFR gezet (371AH), en dan wordt de overeenkomstige string gemaakt (6607H).

adres 6604H

Deze routine wordt door de Factor Evaluator gebruikt om de "STR$"-functie toe te passen op een operand in DAC. Eerst wordt het getal in tekstvorm omgezet en in FBUFFR gezet (3425H) en daarna geanalyseerd om de lengte en het adres ervan te bepalen (6635H). Na een controle of er nog voldoende geheugenruimte over is (668EH) wordt de string naar het stringgeheugen gecopieerd (67C7H) en het signalement van het resultaat opgemaakt (6654H).

adres 6627H

Deze routine controleert eerst of er in het stringgeheugen nog voldoende ruimte over is voor de string waarvan de lengte in register A staat (668EH). Daarna copieert ze de lengte en het adres waar de string zal staan in het stringgeheugen, in DSCTMP.

adres 6636H

Deze routine wordt door de Factor Evaluator gebruikt om de karakterstring te analyseren waarvan het adres in registerpaar HL staat. De string wordt uitgelezen tot een eind-karakter (00H of ") wordt gevonden. Dan worden de lengte en het startadres in DSCTMP gezet (662AH) en de controle loopt door in de routine die het signalement opstelt (6654H).

adres 6654H

Deze routine wordt door de stringfuncties gebruikt om een signalement op te stellen. Het wordt gecopieerd uit DSCTMP naar de eerstvolgende beschikbare plaats in TEMPST, en het adres ervan wordt in DAC gezet. Tenzij TEMPST vol is, wat de foutmelding "String formula too complex" tot gevolg heeft, wordt TEMPPT met drie verhoogd, en de routine stopt.

adres 6678H

Deze routine drukt de tekst of de string af, waarvan het adres in registerpaar HL staat. De string wordt geanalyseerd (6635H) en de ruimte waar hij stond, leeggemaakt (67D3H). De opeenvolgende karakters van de string worden dan afgedrukt, via de standaardroutine OUTDO, tot en met het einde van de string.

adres 668EH

Deze routine controleert of er in het stringgeheugen ruimte is voor de string waarvan de lengte in register A staat. Op het einde van de routine bevat registerpaar DE het adres van de string, waar deze in het stringgeheugen kan staan . Eerst wordt de lengte van de string afgetrokken van het eerste vrije adres,

dat in FRETOP staat. Het resultaat wordt vergeleken met STKTOP, dat het laagste mogelijke adres van het stringgeheugen bevat, om te bepalen of er ruimte vrij is voor de string. Zo ja, wordt FRETOP op de nieuwe waarde gezet en de routine stopt. Is er te weinig ruimte vrij, dan wordt een "garbage collection" gestart (66B6H), om eventuele ongebruikte strings weg te halen. Is er daarna nog steeds niet voldoende ruimte, dan wordt de foutmelding "Out of string space" gegeven.

adres 66B6H

Deze routine haalt de niet langer gebruikte string-inhouden uit het stringgeheugen. Dit proces staat bekend als "garbage collection", letterlijk "ophalen van vuilnis". Het fundamentele probleem met strings is dat, in tegenstelling tot numerieke variabelen, hun lengte niet constant is. Indien de inhoud van strings samen met hun signalement in het variabelengeheugen werden opgeslagen, dan veroorzaakte een eenvoudig statement zoals A$ = A$ + "X" een verschuiving van duizenden bytes in het geheugen, waardoor de loopsnelheid van een programma aanzienlijk vertraagd zou worden. Om dit probleem te omzeilen, slaat de Interpreter de inhoud van strings op een andere plaats op dan de string-variabelen : de tekst wordt opgeslagen in het stringgeheugen, en een signalement van drie bytes in de string-variabele, in het variabelengeheugen. Dit signalement bevat drie bytes, waarin de lengte en het adres van de bijbehorende string zit. Wordt een reeks karakters aan een stringvariabele toegewezen, dan wordt die string gewoon aan het einde van alle bestaande strings in het stringgeheugen toegevoegd, en het signalement van de stringvariabele wordt aangepast. Er wordt niets gedaan om een eventuele vorige inhoud van die stringvariabele weg te halen, door de inhoud van het stringgeheugen te herstructureren : dit zou de tijdswinst weer teniet doen.

Na een aantal toewijzingen van stringvariabelen, is het onvermijdelijk dat het stringgeheugen vol raakt. In een doorsnee programma zullen vele van de opgeslagen strings niet meer gebruikt worden, namelijk de vroegere inhouden van bestaande strings. "Garbage collection" is het proces waardoor die vroegere inhouden weggehaald worden. Elke stringvariabele in het geheugen, met inbegrip van arrays en de plaatselijke variabelen die opduiken tijdens de evaluatie van de zelf-gedefinieerde functies, wordt bekeken tot degene gevonden wordt waarvan de inhoud bovenaan in het stringgeheugen is opgeslagen. Die string wordt dan verzet naar de top van het stringgeheugen, en de inhoud van de variabele wordt aan die plaats aangepast. Vervolgens wordt de volgende string gezocht, en dit proces wordt herhaald totdat elke string die nog bij een signalement hoort, verwerkt is.

Wanneer veel variabelen in het geheugen aanwezig zijn, kan de "garbage collection" een aanzienlijke tijd in beslag nemen. Het hele proces kan gezien worden, door het volgende programma uit te laten voeren, dat herhaaldelijk de string "AAAA" toewijst aan elk element uit het array A$. Het programma loopt tegen volle snelheid tijdens de eerste tweehonderdvijftig toewijzingen. Daarna schort de interpreter de verwerking van het programma op gedurende de tijd die nodig is om de vijftig niet langer gebruikte strings weg te halen. Daarna kunnen weer vijftig

strings worden toegewezen alvorens opnieuw een "garbage collection" nodig is :

```
10 CLEAR 1000
20 DIM A$(200)
30 FOR N=0 TO 200
40 A$(N)= STRING$(4,"A")
50 PRINT ".";
60 NEXT N
70 GOTO 30
```

Het stringgeheugen wordt ook gebruikt om strings in op te slaan die geproduceerd worden tijdens een of andere bewerking. Vele stringfuncties hebben meer dan een argument ("MID$" heeft er drie, bijvoorbeeld) zodat het beheer van de tussenresultaten voor grote problemen kan zorgen. De Interpreter lost dit op, door alle string-resultaten op dezelfde manier aan te pakken. Wanneer een string wordt geproduceerd, wordt de inhoud ervan gewoon bij de strings gevoegd in het stringgeheugen, het signalement wordt bij de bestaande signalementen in TEMPST gevoegd, en het adres ervan in DAC gezet. Het wordt aan de gebruiker van het resultaat overgelaten om die opslagruimte leeg te maken (67D0H) nadat de string verwerkt is. Deze regel geldt voor alle bewerkingen, vanaf de individuele functie-routines, via de Expression evaluator tot en met de routines die de statements verwerken. Er zijn slechts twee uitzonderingen.

De eerste uitzondering wordt gemaakt wanneer de Factor Evaluator in de programmatekst een expliciete string leest, bijvoorbeeld "EEN STRING". In dit geval is het niet nodig om de stringinhoud naar het stringgeheugen te copiëren, het origineel volstaat.

De tweede uitzondering wordt gemaakt indien de Factor Evaluator een referentie naar een variabele vindt. Dan is het onnodig om het signalement in TEMPST te plaatsen : het staat al in het variabelengeheugen, in de bedoelde variabele.

adres 67B7H

Deze routine wordt door de Expression Evaluator gebruikt om twee string-operands samen te voegen. De controle wordt naar dit punt overgedragen, indien na een string-operand een "+"-token wordt gelezen. Daarom wordt allereerst de tweede stringoperand gelezen, via de Factor Evaluator (4DC7H). Uit beide signalementen wordt de lengte gelezen en opgeteld, om de lengte van het resultaat te kennen. Is die lengte groter dan 255, dan wordt de foutmelding "String too long" gegeven. Na een controle of er in het stringgeheugen voldoende ruimte is (6627H), wordt de opslagruimte van beide operands leeggemaakt (67D6H). Dan wordt de eerste string in het stringgeheugen geplaatst (67BFH), gevolgd door de tweede (67BFH). Het signalement van het resultaat wordt opgesteld (6654H) en de controle gaat terug naar de Expression Evaluator (4C73H).

adres 67D0H

Deze routine maakt de opslagruimte beschikbaar die wordt ingenomen door een string waarvan het adres van het signalement in DAC staat. Het adres van het signalement wordt uit DAC gelezen en vergeleken met dat van het laatste signalement in

TEMPSI (67EEH). Zijn beide verschillend, dan stopt de routine. Zijn ze gelijk, dan wordt TEMPST drie bytes kleiner gemaakt, waardoor het laatste signalement eruit verdwijnt. Uit het signalement wordt het adres van de string-inhoud gelezen, en dat wordt vergeleken met FRETOP. Staat de string niet op de laagste plaats in het stringgeheugen, dan stopt de routine. Is dat wel zo, dan wordt de lengte van de string bij FRETOP opgeteld, en die waarde wordt vervolgens in FRETOP geladen. Daardoor is de opslagruimte die de string in beslag nam, weer beschikbaar.

adres 67FFH

Deze routine wordt door de Factor Evaluator gebruikt om de "LEN"-functie toe te passen op een operand in DAC. De ruimte die de operand in beslag neemt, wordt vrijgemaakt (67D0H) en de stringlengte wordt uit het signalement gehaald, en als een geheel getal in DAC gezet (4FCFH).

adres 680BH

Deze routine wordt door de Factor Evaluator gebruikt om de "ASC"-functie toe te passen op een operand in DAC. De ruimte die de operand in beslag neemt, wordt vrijgemaakt en de lengte van de string wordt bekeken (6803H). Is die nul, dan wordt de foutmelding "Illegal function call" gegeven (475AH). Zoniet wordt het eerste karakter uit de string gehaald en als een geheel getal in DAC gezet (4FCFH).

adres 681BH

Deze routine wordt door de Factor Evaluator gebruikt om de "CHR$"-functie toe te passen op een operand in DAC. Eerst wordt gecontroleerd of voldoende ruimte beschikbaar is (6625H) en dan wordt de operand in een geheel getal van één byte omgezet (521FH). Dit karakter wordt in het stringgeheugen gezet en er wordt een signalement opgemaakt (6654H).

adres 6829H

Deze routine wordt door de Factor Evaluator gebruikt om de "STRING$"-functie toe te passen. Eerst wordt gecontroleerd of er een "(" volgt na het token. Dan wordt de operand die de lengte aangeeft, geëvalueerd en in register E geladen (521CH). Daarna wordt de tweede operand geëvalueerd (4C64H). Is die numeriek, dan wordt hij omgezet in een geheel getal van één byte (521FH) en in register A geladen. Is het een string, dan wordt het eerste karakter ervan in register A geladen (680FH). Daarna loopt de controle door in de "SPACE$"-routine, waar de string wordt geproduceerd.

adres 6848H

Deze routine wordt door de Factor Evaluator gebruikt om de "SPACE$"-functie toe te passen op een operand in DAC. Eerst wordt die operand omgezet in een geheel getal van één byte en in register E geladen (521FH). Dan wordt gecontroleerd of nog voldoende ruimte vrij is in het stringgeheugen (6627H) en vervolgens wordt het gewenste aantal spaties in het stringgeheugen gezet. Daarna wordt een signalement opgesteld (6654H).

adres 6861H

Deze routine wordt door de Factor Evaluator gebruikt om de "LEFT$"-functie toe te passen. Het adres van het signalement van de eerste operand en de tweede operand (een geheel getal) worden op de stack van de Z80 verwacht. Deze laatste wordt van de stack gehaald (68E3H) en vergeleken met de lengte van de oorspronkelijke string. Is die kleiner dan de tweede operand, dan komt de oorspronkelijke lengte in de plaats van die operand. Dan wordt er gecontroleerd of er voldoende ruimte is in het stringgeheugen (668EH). Is dat zo, dan worden het gewenste aantal karakters uit de oorspronkelijke string naar dat stringgeheugen gecopieerd, te beginnen bij de start van de oorspronkelijke string (67C7H). De opslagruimte van de oorspronkelijke string wordt vrijgemaakt (67D7H) en het signalement van het resultaat opgesteld (6654H).

adres 6891H

Deze routine wordt door de Factor Evaluator gebruikt om de "RIGHT$"-functie toe te passen. Het adres van het signalement van de eerste operand en de tweede operand (een geheel getal) worden op de stack van de Z80 verwacht. De lengte van de deelstring wordt van de stack gehaald (68E3H) en afgetrokken van de lengte van de oorspronkelijke string. Daardoor is het begin van de deelstring bekend. Nu gaat de controle over naar de "LEFT$"-routine, waar de deelstring wordt gevormd (6865H).

adres 689AH

Deze routine wordt door de Factor Evaluator gebruikt om de "MID$"-functie toe te passen. Het adres van het signalement van de eerste operand en de tweede operand (een geheel getal) worden op de stack van de Z80 verwacht. Het beginpunt wordt van de stack gehaald (68E8H) en gecontroleerd. Is dat nul, dan wordt de foutmelding "Illegal function call" gegeven (475AH). Daarna wordt de facultatieve operand die de lengte van de deelstring aangeeft, geëvalueerd (69E4H), waarna de controle overgaat naar de "LEFT$"-routine, waar de deelstring wordt gevormd (6869H).

adres 68BBH

Deze routine wordt door de Factor Evaluator gebruikt om de "VAL"-functie toe te passen op een operand in DAC. De lengte van de string wordt uit het signalement gehaald (6803H) en gecontroleerd. Is ze nul, dan wordt dat als een geheel getal in DAC gezet (4FCFH). Dan wordt de lengte opgeteld bij het startadres van de string-inhoud, om het karakter te localiseren dat direkt op de string volgt. Dit karakter wordt tijdelijk vervangen door een nul-byte, en daarna wordt de string omgezet in een getal en in DAC gezet (3299H). Nu wordt het nul-byte opnieuw vervangen door het oorspronkelijke karakter, en de routine stopt. Het nul-byte dient tijdelijk als stop-teken : strings zitten direkt na elkaar in het stringgeheugen, en zonder het stop-byte zou de omzetroutine gewoon doorgaan met de volgende string.

adres 68E3H

Deze routine wordt gebruikt door de "LEFT$"-, "MID$"- en "RIGHT$"-routines, om te controleren of het eerstvolgende karakter in de programmatekst een ")" is, en daarna om een operand van de stack van de Z80 in registerpaar DE te POPpen.

adres 68EBH

Deze routine wordt door de Factor Evaluator gebruikt om de "INSTR"-functie toe te passen. De eerste operand, die zowel de startpositie als de te doorzoeken string kan zijn, wordt geëvalueerd (4C62H) en het type ervan wordt gecontroleerd. Is de operand de te doorzoeken string, dan wordt als standaard een startpositie één aangenomen. Is de eerste operand de startpositie, dan wordt de waarde ervan gecontroleerd, en dan wordt de string-operand geëvalueerd (4C64H). Nu wordt de operand die de te zoeken string bevat, geëvalueerd (4C64H) en de opslagruimte van beide strings vrijgemaakt (67D0H). De lengte van de te zoeken string wordt gecontroleerd en indien die nul is, wordt de startpositie in DAC gezet (4FCFH). Dan wordt, te beginnen bij de startpositie, de te zoeken string vergeleken met opeenvolgende karakters in de te doorzoeken string tot ofwel een gelijke groep wordt gevonden, ofwel het einde van de te doorzoeken string bereikt is. In dit laatste geval wordt nul als een geheel getal in DAC gezet (4FCFH), in het eerste geval de positie van het eerste karakter van de deelstring in de te doorzoeken string.

adres 696EH

Deze routine verwerkt het "MID$"-statement. Na een controle of het "("-karakter er achter staat, wordt de doelstring opgezocht (5EA4H) en gecontroleerd of het wel een string is (3058H). Uit de variabele wordt nu het adres van de stringinhoud gelezen, en dit wordt gecontroleerd, om te zien of de string in het programma staat, wat het geval is indien de string expliciet opgegeven is. Is dat zo, dan wordt de inhoud naar het stringgeheugen gecopieerd (6611H) en in de variabele wordt een nieuw signalement gezet (2EF3H). Daarmee wordt vermeden dat de tekst van het programma gewijzigd wordt. Nu wordt de startpositie geëvalueerd (521CH) en gecontroleerd. Is ze nul, dan wordt de foutmelding "Illegal function call" gegeven (475AH). De facultatieve operand die de lengte van de deelstring aangeeft, wordt geëvalueerd (69E4H), en daarna de string die de oude inhoud moet vervangen (4C5FH). De opslagruimte van deze laatste wordt daarna vrijgemaakt (67D0H). Vervolgens worden opeenvolgende karakters uit de vervangstring gecopieerd naar de doelstring, tot ofwel de opgegeven lengte bereikt is, ofwel de vervangstring ten einde is.

adres 69E4H

Deze routine wordt door diverse stringfuncties gebruikt om een facultatieve operand te evalueren (521CH), en het resultaat in register E te zetten. Wordt geen operand gevonden, dan wordt de standaardwaarde 255 in register E geladen.

adres 69F2H

Deze routine wordt door de Factor Evaluator gebruikt om de "FRE"-functie toe te passen op een operand in DAC. Is die numeriek, dan wordt het verschil, met enkele precisie, tussen de inhoud van de Stack Pointer van de Z80 en de inhoud van STREND in DAC gezet (4FC1H). Is de operand een string, dan wordt diens opslagruimte vrijgemaakt (67D3H) en er wordt een "garbage collection" op gang gebracht (66B6H). Daarna wordt het verschil, met enkele precisie, tussen de inhoud van FRETOP en van STKTOP in DAC gezet (4FC1H).

adres 6A0EH

Deze routine wordt gebruikt door de routines die In/Out-bewerkingen met files verzorgen, om een specificatie zoals "A:NAAM.BAS" te analyseren. De specificatie bestaat uit drie delen: het apparaat, de naam, en de type-aanduiding. Bij de start van de routine bevat registerpaar HL het startadres van de specificatie in de programmatekst. Op het einde bevat register D de code van het apparaat, staat de naam van de file op posities 0 tot 7 van FILNAM en de type-aanduiding op posities 8 tot 10. Ongebruikte posities worden met spaties opgevuld.

De string die de specificatie bevat wordt geëvalueerd (4C64H), en zijn opslagruimte vrijgemaakt (67D0H). Is de lengte ervan nul, dan wordt de foutmelding "Bad file name" geproduceerd (6E6BH). De naam van het apparaat wordt eerst ontleed (6F15H). Nu worden opeenvolgende karakters uit de string gehaald en in FILNAM gezet totdat ofwel het einde van de string bereikt is, ofwel een "." wordt gevonden, ofwel FILNAM vol is. Indien de string een controlekarakter bevat (code lager van 20H) wordt de foutmelding "Bad file name" gegeven (6E6BH). Indien de specificatie een type-aanduiding bevat wordt eveneens de foutmelding "Bad file name" gegeven (6E6BH) als deze laatste langer is dan drie karakters of als de naam van de file langer is dan acht karakters. Bevat de specificatie geen type-aanduiding, dan is de lengte van de naam van geen belang : overtollige karakters worden genegeerd.

adres 6A6DH

Deze routine wordt gebruikt door de routines die In/Out-bewerkingen met files verzorgen, om het controleblok van de In/Out-buffer te localiseren, waarvan het nummer in register A staat. Eerst wordt het buffernummer met MAXFIL vergeleken, en indien het nummer te groot is, wordt de foutmelding "Bad file number" gegeven (6E7DH). Zoniet, wordt het overeenkomstige adres gelezen in het blok dat de file-pointers bevat, en in registerpaar HL gezet. De manier waarop de buffer werkt, wordt uit byte 0 van het controleblok gelezen en in register A gezet.

adres 6A9EH

Deze routine wordt gebruikt door de routines die In/Out-bewerkingen met files verzorgen, om een In/Out-buffernummer te evalueren en het overeenkomstige controleblok te localiseren. Een "#"-karakter wordt overgeslagen (4666H) en dan wordt het buffernummer geëvalueerd (521CH). Het controleblok wordt gelocaliseerd (6A6DH) en indien het Mode-byte van de buffer nul

is, wordt de foutmelding "File not open" gegeven. In de andere gevallen wordt het adres van het controleblok in PTRFIL gezet, om de output van de Interpreter in andere banen te leiden.

adres 6AB7H

Deze routine verwerkt het "OPEN"-statement. De specificatie van de file wordt geanalyseerd (6A0EH), en een daaropvolgende aanduiding van mode wordt omgezet in een overeenkomstige code. Die codes zijn : 01H ("FOR INPUT"), 02H ("FOR OUTPUT"), en 08H ("FOR APPEND"). Wordt geen van deze aanduidingen gegeven, dan wordt code 04H (random file) aangenomen. Er wordt gecontroleerd of de karakters "AS" er staan, en het buffernummer wordt geëvalueerd (521CH). Is dat nul, dan wordt de foutmelding "Bad file number" gegeven (6E7DH). Zoniet wordt het controleblok gelocaliseerd (6A6DH) en indien het modebyte van de buffer iets anders dan nul bevat, wordt de foutmelding "File already open" gegeven (6E6EH). De code van het apparaat wordt in byte 4 van het controleblok geladen, de open-functie wordt afgehandeld (6F8FH) en de output van de Interpreter wordt terug naar het scherm geleid (4AFFH).

adres 6B24H

Deze routine wordt gebruikt door de routines die In/Out-bewerkingen met files verzorgen, om de In/Out-buffer waarvan het nummer in register A staat, te sluiten. Het controleblok wordt gelocaliseerd (6A6DH) en indien de buffer in gebruik is, wordt de CLOSE-functie afgehandeld (6F8FH). Daarna wordt de buffer met nullen gevuld (6CEAH). PTRFIL en het mode byte van het controleblok worden op nul gezet, waardoor de output van de Interpreter terug naar het scherm wordt geleid.

adres 6B5BH

Deze routine verwerkt de statements "LOAD", "MERGE" en "RUN file-specificatie". De specificatie wordt geanalyseerd (6A0EH) en dan wordt, enkel voor "LOAD" en "RUN", in de programmatekst gekeken of de auto-run optie "R" gespecificeerd werd. In/Out-buffer nummer 0 wordt geopend als input-buffer (6AFAH) en het eerste byte van FILNAM wordt op FFH gezet indien het programma automatisch moet starten. Enkel indien het om "LOAD" of "RUN" gaat, wordt eventuele aanwezige programmatekst uit het geheugen gewist via de "NEW"-routine (6287H). Omdat daardoor de output van de Interpreter opnieuw naar het scherm wordt gestuurd, wordt het controleblok opnieuw gelocaliseerd en in PTRFIL gezet (6AAAH). Daarna gaat de controle over naar de Hoofdlus van de Interpreter (4134H), waar de programmatekst in het geheugen geladen wordt alsof het commando direkt werd ingetypt. Merk op dat geen enkele foutcontrole wordt uitgevoerd op de data die worden ingelezen.

adres 6BA3H

Deze routine verwerkt het "SAVE"-statement. De file-specificatie wordt geanalyseerd (6A0EH) en daarna wordt in de programmatekst gekeken of het achtervoegsel "A" (ASCII) aanwezig is. Dit heeft enkel belang bij disk-operaties, en maakt op een standaard MSX-machine niets uit. In/Out-buffer 0 wordt geopend als output-kanaal (6AFAH) en de controle wordt overgedragen aan de "LIST"-

routine (522EH), waar de programmatekst naar buiten wordt gestuurd. Merk op, dat dit met geen enkele foutcontrole gepaard gaat.

adres 6BDAH

Deze routine wordt gebruikt door de routines die In/Out-bewerkingen met files uitvoeren, om de code op te zoeken van de In/Out-buffer die op dat ogenblik in gebruik is. Uit PTRFIL wordt het adres gelezen van het controleblok en daarna wordt de code van het apparaat uit byte 4 van dit controleblok gelezen en in register A geladen.

adres 6BE7H

Deze routine wordt gebruikt door de routines die In/Out-bewerkingen met files uitvoeren, om een bewerking uit te voeren op een aantal In/Out-buffers. Het adres van de uit te voeren routine staat in registerpaar BC en het aantal buffers in register A. Indien bijvoorbeeld in registerpaar BC, 6B24H stond en in register A, 03H, dan zou dat betekenen dat buffers 3,2,1 en Ø gesloten worden. De routine voert een iets verschillende bewerking uit indien bij de start vlag Z gereset is. In dat geval worden de nummers van de In/Out-buffers achtereenvolgens uit de programmatekst gehaald en geëvalueerd (521CH) voordat de juiste bewerking wordt uitgevoerd. Een kenmerkend voorbeeld zou zijn : "#1,#2".

adres 6C14H

Deze routine verwerkt het "CLOSE"-statement. In registerpaar BC wordt 6B24H geladen, in register A de inhoud van MAXFIL en het nodige aantal buffers wordt gesloten (6BE7H).

adres 6C1CH

Deze routine wordt gebruikt door de routines die In/Out-bewerkingen uitvoeren met files, om elke In/Out-buffer te sluiten. In registerpaar BC wordt 6B24H geladen, in register A de inhoud van MAXFIL en alle buffers worden gesloten (6BE7H).

adres 6C2AH

Deze routine verwerkt het "LFILES"-statement. PRTFLG wordt op een van nul verschillende waarde gezet, om de output naar de printer te leiden, en de controle loopt door in de "FILES"-routine.

adres 6C2FH

Deze routine verwerkt het "FILES"-statement. Op een standaard MSX-machine (zonder disk-drive; de vertaler) wordt de foutmelding "Illegal function call" gegeven (475AH).

adres 6C35H

De "PUT"- en "GET"-routines (7758H) geven de controle over aan deze routine indien in de programmatekst iets anders dan een "SPRITE"-token volgt. Op een standaard MSX-machine wordt de foutmelding "Sequential I/O only" gegeven (6E86H).

adres 6C48H

Deze routine wordt gebruikt door de routines die In/Out-bewerkingen uitvoeren met files, om het karakter dat in register A staat, sequentieel naar buiten te sturen. Het karakter wordt in register C geladen en de sequentiële outputfunctie wordt afgehandeld (6F8FH).

adres 6C71H

Deze routine wordt gebruikt door de routines die In/Out-bewerkingen uitvoeren met files, om een karakter sequentieel in te lezen. De sequentiële input-routine wordt afgehandeld (6F8FH) en het karakter wordt in register A teruggebracht. Vlag C geeft een End of File aan.

adres 6C87H

Deze routine wordt door de Factor Evaluator gebruikt om de "INPUT$"-functie toe te passen. Er wordt gecontroleerd of het "$"-teken en een "(" in de programmatekst staan, en dan wordt de operand die de lengte aangeeft, geëvalueerd (521CH). Is er een nummer van een In/Out-buffer aanwezig, dan wordt dat geëvalueerd, het controleblok wordt gelocaliseerd (6A9EH) en het mode-byte wordt onderzocht. Staat de buffer niet in input- of random-mode, dan wordt de foutmelding "Input past end" gegeven (6E83H). Na een controle of er voldoende ruimte is (6627H) wordt het vereiste aantal karakters sequentieel ingelezen (6C71H) of via de standaardroutine CHGET opgehaald, en in het stringgeheugen gecopieerd. Tenslotte wordt het signalement van het resultaat opgesteld (6654H).

adres 6CEAH

Deze routine wordt gebruikt door de routines die In/Out-bewerkingen met files uitvoeren. PTRFIL bevat het adres van een controleblok van een buffer : die buffer wordt met 256 nullen gevuld.

adres 6CFBH

Deze routine wordt gebruikt door de routines die In/Out-bewerkingen met files uitvoeren. In PTRFIL staat het adres van een controleblok van een buffer. Het startadres van die buffer wordt in registerpaar HL berekend, wat neerkomt op negen optellen bij het adres van het controleblok.

adres 6D03H

Deze routine wordt door de Factor Evaluator gebruikt om de "LOC"-functie toe te passen op de In/Out-buffer waarvan het nummer in DAC staat. Het controleblok wordt gelocaliseerd (6A6AH) en de functie wordt afgehandeld (6F8FH). Op een standaard MSX-machine wordt de foutmelding "Illegal function call" gegeven (475AH).

adres 6D14H

Deze routine wordt door de Factor Evaluator gebruikt om de "LOF"-functie toe te passen op de In/Out-buffer waarvan het

nummer in DAC staat. Het controleblok wordt gelocaliseerd (6A6AH) en de functie wordt afgehandeld (6F8FH). Op een standaard MSX-machine wordt de foutmelding "Illegal function call" gegeven (475AH).

adres 6D25H

Deze routine wordt door de Factor Evaluator gebruikt om de "EOF"-functie toe te passen op de In/Out-buffer waarvan het nummer in DAC staat. Het controleblok wordt gelocaliseerd (6A6AH) en de functie wordt afgehandeld (6F8FH).

adres 6D39H

Deze routine wordt door de Factor Evaluator gebruikt om de "FPOS"-functie toe te passen op de In/Out-buffer waarvan het nummer in DAC staat. Het controleblok wordt gelocaliseerd (6A6AH) en de functie wordt afgehandeld (6F8FH). Op een standaard MSX-machine wordt de foutmelding "Illegal function call" gegeven (475AH).

adres 6D48H

Deze routine krijgt de controle, wanneer de Hoofdlus van de Interpreter een direct statement vindt, dat wil zeggen een statement zonder voorafgaand regelnummer. Eerst wordt via de standaardroutine ISFLIO bepaald of een "LOAD"-statement wordt uitgevoerd. Blijkt de input vanaf het toetsenbord te komen, dan wordt de controle overgedragen naar het uitvoeringspunt van de Verwerkingslus (4640H) waar het statement wordt uitgevoerd. Komt de input van de cassette, dan wordt buffer nummer 0 gesloten (6B24H) en wordt de foutmelding "Direct statement in file" gegeven (6E71H). Op een standaard MSX-machine zou dit kunnen voorkomen hetzij door een fout in de cassette, hetzij bij een poging om een tekstfile zonder regelnummers in te laden.

adres 6D57H

Deze routine wordt gebruikt door de routines die "INPUT", "LINE INPUT" en "PRINT verwerken, om te controleren of er in de programmatekst een "#"-teken staat. Is dat zo, dan wordt het nummer van de In/Out-buffer geëvalueerd (521CH), het controleblok gelocaliseerd en het adres daarvan in PTRFIL gezet (6AAAH). Dan wordt het mode-byte van het controleblok vergeleken met het getal dat door een van de routines in register C werd geladen. Verschillen ze, dan wordt de foutmelding "Bad file number" gegeven (6E7DH). Voor het "PRINT"-statement zijn de modes "output", "random" en "append" toegelaten. Bij "INPUT" en "LINE INPUT" zijn enkel "input" en "random"-modes toegelaten. Merk wel op dat bij een standaard MSX-machine niet al die modes ook op lagere niveaus toegelaten zijn. Dat houdt in dat in een later stadium van de verwerking een of andere foutmelding zal gegeven worden indien een bepaalde mode niet door de controles komt.

adres 6D83H

Deze routine wordt gebruikt door het "INPUT"-statement, om een string uit een In/Out-buffer te lezen. Eerst wordt een RETURN-adres klaargezet naar de "READ/INPUT"-routine (4BF1H). De

karakters die de input-string begrenzen - een komma en een spatie voor een numerieke variabele en een komma alleen voor een string - worden in registers D en E geladen, en de controle gaat over naar de "LINE INPUT"-routine (6DA3H).

adres 6D8FH

Deze routine verwerkt het "LINE INPUT"-statement wanneer de input uit een In/Out-buffer komt. Het buffernummer wordt geëvalueerd, het controleblok gelocaliseerd en de mode gecontroleerd (6D55H). Dan wordt de variabele gelocaliseerd (5EA4H) waar de input aan toegewezen dient te worden, en zijn type wordt gecontroleerd, om zeker te zijn dat het een string is (3058H). Een RETURN-adres naar de "LET"-routine wordt klaargezet (487BH), waar de toewijzing zal gebeuren, en de string wordt ingelezen.

De karakters worden sequentieel ingelezen (6C71H) en in BUF gezet, tot het correcte eind-karakter gelezen wordt, of een EOF wordt ontdekt of BUF vol is (6E41H). Is het inlezen beëindigd, en de input moet in een numerieke variabele gezet worden, dan wordt de string in een getal omgezet en in DAC geplaatst (3299H). Wordt de input aan een stringvariabele toegewezen, dan wordt hij geanalyseerd en het signalement van het resultaat wordt opgesteld (6638H).

Bij "LINE INPUT" worden alle karakters geaccepteerd, tot een "CR"-code gelezen wordt. Merk op dat, indien deze code wordt voorafgegaan door een "LF"-code, ze niet als begrenzer zal werken maar gewoon als een deel van de string zal ingelezen worden. Bij "INPUT" voor een numerieke variabele worden de spaties er af gehaald, en de dan volgende karakters geaccepteerd tot aan een "CR"-code, een spatie of een komma. Hier geldt ook dat een "CR"-code niet als begrenzer werkt indien ze wordt voorafgegaan door een "LF"-code . In dit geval wordt de "CR" evenwel niet in BUF gezet, maar gewoon genegeerd. Bij "INPUT" in een stringvariabele, worden voorafgaande spaties weggelaten en dan volgende karakters geaccepteerd tot aan een "CR"-code of een komma. Ook hier geldt dat een "CR"-code niet als begrenzer werkt indien ze wordt voorafgegaan door een "LF"-code. In dit geval worden geen van beide codes in BUF geplaatst, maar allebei genegeerd. Anders wordt het indien, na eventuele voorafgaande spaties, een aanhalingsteken gelezen wordt : daarna wordt elk karakter geaccepteerd en in BUF opgeslagen, totdat een tweede aanhalingsteken wordt gelezen.

Wanneer de input-string ingelezen is, wordt aan de hand van het begrenzingsteken bepaald of er nog iets speciaals moet gebeuren met eventuele volgende karakters. Was het laatste teken een aanhalingsteken of een spatie, dan worden eventuele volgende spaties ingelezen en genegeerd, tot een ander teken wordt gevonden. Is dat een komma of een "CR"-code, dan wordt het ingelezen en genegeerd. In de andere gevallen wordt een terugzet-functie uitgevoerd (6F8FH) om het karakter terug naar de In/Out-buffer te sturen. Was het laatste karakter een "CR"-code, dan wordt het volgende karakter ingelezen en gecontroleerd. Is het een "LF"-code, dan wordt het ingelezen maar genegeerd. Is het geen "LF"-code, dan wordt een terugzet-functie uitgevoerd (6F8FH) om het karakter terug in de In/Out-buffer te zetten.

adres 6E6BH TABEL

Op dit adres staat een groep van tien foutcodes die verband houden met In/Out-bewerkingen met files. Register E wordt met de juiste foutcode geladen en de controle wordt overgedragen aan de foutverwerker (406FH) :

ADRES	FOUTMELDING
6E6BH	Bad file name
6E6EH	File already open
6E71H	Direct statement in file
6E74H	File not found
6E77H	File not open
6E7AH	Field overflow
6E7DH	Bad file number
6E80H	Internal error
6E83H	Input past end
6E86H	Sequential I/O only

adres 6E92H

Deze routine verwerkt het "BSAVE"-statement. De specificatie wordt geanalyseerd (6A0EH) en het startadres geëvalueerd (6F0BH). Dan wordt het eindadres geëvalueerd (6F0BH) en in SAVEND gezet, gevolgd door het facultatieve opstartadres (6F0BH), dat in SAVENT gezet wordt. Wordt geen opstartadres opgegeven, dan wordt in de plaats daarvan het startadres gezet. De code van het apparaat wordt gecontroleerd, om zeker te zijn dat het om "CAS" gaat. Is dat niet zo, dan wordt de foutmelding "Bad file name" gegeven (6E6BH). Zo ja, worden de gegevens naar de cassette geschreven (6FD7H). Merk op dat die niet gebufferd worden, maar rechtstreeks geschreven worden. Er wordt geen informatie meegestuurd, die foutcontrole mogelijk kan maken.

adres 6EC6H

Deze routine verwerkt het "BLOAD"-statement. De specificatie wordt geanalyseerd (6A0EH) en indien de auto-run optie "R" in de programmatekst staat, wordt RUNBNF verschillend van nul gemaakt. De facultatieve verschuiving van het startadres, met standaardwaarde nul, wordt nu geëvalueerd (6F0BH) en er wordt gecontroleerd of de code als apparaat "CAS:" aangeeft. Is dit niet zo, dan wordt de foutmelding "Bad file name" gegeven (6E6BH). Vervolgens worden de gegevens rechtstreeks van de cassette gelezen (7014H), zonder buffering en foutcontrole.

adres 6EF4H

Deze routine wordt aangesproken nadat de "BLOAD"-routine alle gegevens in het geheugen heeft geladen. Indien RUNBNF nul is, wordt buffer nummer 0 gesloten (6B24H) en de controle gaat over naar de Verwerkingslus. Zoniet, wordt buffer nummer 0 gesloten (6B24H), een RETURN-adres 6CF3H wordt klaargezet (daar wordt het adres in de programmatekst waar de Interpreter aan het werk was, in registerpaar HL gePOPt, en de Verwerkingslus neemt over) en de controle gaat over op het adres dat in SAVENT staat.

adres 6F0BH

Deze routine wordt gebruikt door de "BLOAD" en "BSAVE"-routines om een operand te evalueren die een adres bevat. Het resultaat wordt in registerpaar DE gezet. Eerst wordt de operand geëvalueerd (4C64H), en dan in een geheel getal omgezet (5439H).

adres 6F15H

Deze routine wordt gebruikt door de routine die de file-specificatie analyseert, om de naam van een apparaat, zoals "CAS:", te ontleden. Bij de start bevat registerpaar HL het startadres van de string die de specificatie bevat, en register E de lengte ervan. Bevat die string geen naam van een apparaat, dan wordt de standaardcode ("CAS:"=FFH) in register A geladen, en vlag Z geset. Staat er wel een toegestane naam in, dan wordt de overeenkomstige code in register A geladen en vlag Z gereset.

De specificatie wordt doorzocht tot een ":" wordt gevonden. De zo gevonden naam wordt vergeleken met elk van de toegestane namen in de tabel op adres 6F76H. Wordt de naam gevonden, dan laadt de routine de overeenkomstige code in de tabel, in register A. Staat de gevonden naam niet in de tabel, dan gaat de controle over naar de routine die externe ROM uitleest (55F8H). Kleine letters in de opgegeven naam worden hoofdletters gemaakt, om de vergelijkingen gemakkelijker te maken. Dat wil zeggen dat "crt" en "CRT" beide naar hetzelfde apparaat verwijzen.

adres 6F76H TABEL

De tabel op dit adres wordt gebruikt door de routine die de naam van een apparaat ontleedt. De tabel bevat de vier namen en overeenkomstige codes, die op een standaard MSX-machine toegestaan worden :

CAS ... FFH   LPT ... FEH   CRT ... FDH   GRP ... FCH

adres 6F87H TABEL

De tabel op dit adres wordt gebruikt door de routine die de functies afhandelt (6F8FH). Hij bevat het adres van de tabel die de functies decodeert, voor elk van de vier apparaten die standaard in MSX voorkomen :

CAS ... 71C7H   LPT ... 72A6H   CRT ... 71A2H   GRP ... 71B2H

adres 6F8FH

Deze routine regelt de afhandeling van de In/Out-bewerkingen met files. Verderop in het boek wordt ze "de dispatcher" genoemd. Samen met de bufferstructuur van de Interpreter geeft ze de beschikking over een samenhangend systeem om, onafhankelijk van het te bedienen apparaat, binnenkomende en uitgaande gegevens te verwerken. De vereiste functiecode wordt in register A aangebracht, en het adres van het controleblok van de buffer in registerpaar HL.

De code van het apparaat wordt uit byte 4 van het controleblok gelezen. Er wordt gecontroleerd of het één van de vier standaard apparaten is. Zoniet, wordt de controle overgedragen aan de

gelijkwaardige routine in externe ROM (564AH). Was het wel een standaard-apparaat, dan wordt uit de tabel op adres 6F8FH het adres gelezen van de tabel waar de functies van dat apparaat gedecodeerd worden. Uit die laatste tabel wordt het adres gelezen van de gewenste functie, en de controle gaat over naar de juiste functie-routine.

adres 6FB7H

Deze routine verwerkt het "CSAVE"-statement. De naam van de file wordt geëvalueerd (7098H), gevolgd door de facultatieve baud-rate (7A2DH). Dan wordt het identificatieblok naar de cassette geschreven (7125H) met in het byte dat het filetype aangeeft, de waarde D3H. De inhoud van het programmageheugen wordt rechtstreeks naar cassette geschreven, als één blok (713EH). Er wordt geen informatie meegestuurd die foutcontrole mogelijk kan maken.

adres 6FD7H

Deze routine neemt de controle over van de "BSAVE"-routine, om een geheugenblok op cassette te schrijven. Eerst wordt het identificatieblok geschreven (7125H), met in het byte dat het filetype aangeeft, de waarde D0H. De motor wordt aangezet en er wordt een kort aanloopblok ("header") geschreven (72F8H). Het startadres van het blok wordt van de stack van de Z80 gePOPt en weggeschreven in de volgorde : LSB, MSB (7003H). Het eindadres wordt uit SAVEND gehaald en in dezelfde volgorde weggeschreven (7003H). Daarna volgt het opstartadres, dat uit SAVENT wordt gelezen. Vervolgens wordt het omschreven geheugenblok naar cassette geschreven, één byte tegelijk (72DEH), waarna de cassettemotor wordt afgezet via de standaardroutine TAPOOF. Ook hier wordt geen informatie meegegeven om foutcontrole mogelijk te maken.

adres 7003H

Deze routine schrijft de inhoud van registerpaar HL naar de cassette, eerst register L (72DEH) en dan register H (72DEH).

adres 700BH

Deze routine leest twee bytes van een cassette en laadt ze in registerpaar HL. Het eerste byte wordt in register L geladen (72D4H), het tweede in register H (72D4H).

adres 7014H

Deze routine neemt de controle over van de "BLOAD"-routine, om gegevens van cassette in het geheugen te laden. De inhoud van de cassette wordt ingelezen, tot een identificatieblok wordt gevonden dat als type van file, de waarde D0H bevat, en de correcte naam (70B8H). Dan wordt het aanloopblok op de cassette gezocht (72E9H). De adres-verschuiving wordt van de stack van de Z80 gePOPt en opgeteld bij het startadres, dat van de cassette werd gelezen (700BH). Het eindadres wordt van de cassette gelezen (700BH) en ook daar wordt de verschuiving bij opgeteld. Het opstartadres wordt van de cassette gelezen (700BH) en in SAVENT gezet, waar het uitgelezen kan worden indien de code automatisch moet opstarten. Daarna worden opeenvolgende gegevens

van de cassette gelezen (72D4H) en in het geheugen geladen, te beginnen bij het startadres, tot het eindadres wordt bereikt. Tenslotte wordt de motor uitgezet via de standaardroutine TAPIOF, en de controle wordt overgedragen naar het eindpunt van de "BLOAD"-routine (6EF4H).

adres 703FH

Deze routine verwerkt de statements "CLOAD" en "CLOAD?". Eerst wordt gecontroleerd of het "PRINT"-token (91H) in de programmatekst staat, want het "?"-karakter wordt als dusdanig gecodeerd. Dan wordt de filenaam geëvalueerd (708CH). De inhoud van de cassette wordt ingelezen tot een identificatieblok wordt gevonden dat het filetype D3H en de correcte naam (70B8H) bevat. Bij "CLOAD" wordt de "NEW"-routine aangeroepen (6287H) om het programmageheugen leeg te maken. Bij "CLOAD?" worden alle adrespointers in het programmageheugen in regelnummers omgezet (54EAH), om ze te kunnen vergelijken met de gegevens op de cassette.

Nu wordt het aanloopblok op de cassette gezocht en opeenvolgende gegevens worden van de cassette gelezen en in het geheugen geplaatst ofwel vergeleken met de momentele inhoud van het geheugen (715DH). Nadat het gegevensblok volledig werd gelezen, wordt de tekst "Ok" afgedrukt (6678H) en de controle gaat over naar het einde van de Hoofdlus van de Interpreter (4237H) om de pointers van het variabelengeheugen op hun nieuwe waarden te zetten. Bij het "CLOAD?"-statement zal de uitlezing van het gegevensblok stoppen indien het byte dat werd ingelezen van cassette, niet hetzelfde is als het overeenkomstige byte in het geheugen. Gebeurt dit op een adres dat ligt boven het programmageheugen, dan stopt deze routine met het "Ok"-bericht. Zoniet, wordt de foutmelding "Verify error" gegeven.

adres 708CH

Deze routine wordt gebruikt door de "CLOAD" en "CSAVE"-routines om een filenaam in de programmatekst te evalueren. Deze twee statements roepen de routine op verschillende adressen aan, zodat voor "CLOAD" een lege string als naam toegestaan wordt, maar niet voor "CSAVE". De string met de filenaam wordt geëvalueerd (4C64H), zijn opslagruimte vrijgemaakt (680FH) en de eerste zes karakters worden in FILNAM gezet. Wanneer de naam langer is dan zes karakters, worden de overtollige karakters genegeerd. Is de naam korter dan zes karakters, dan wordt FILNAM bijgevuld met spatiès.

adres 70B8H

Deze routine localiseert een identificatieblok op een cassette. Ze wordt gebruikt door de "CLOAD" en "BLOAD"-routines, en maakt ook een deel uit van de open-functie van de routine die de In/Out-bewerkingen afhandelt (6F8FH) indien het apparaat "CAS" is en er in input-mode wordt gewerkt. Bij de start staat de naam van de file in FILNAM. Register C bevat het type van de file : D3H voor een gecodeerde BASIC-file (CLOAD), D0H voor een binaire file (BLOAD) en EAH voor een ASCII-file (LOAD of datafile).

De motor van de cassetterecorder wordt ingeschakeld en de cassette wordt ingelezen tot een aanloopblok wordt gevonden

(72E9H). Voor elk identificatieblok komt een groep van tien karakters die het type van de file aangeven. Daarom worden opeenvolgende karakters van de cassette ingelezen (72D4H) en vergeleken met het gewenste type. Komen de karakters niet overeen, dan keert de routine terug naar het beginpunt en zoekt verder naar het volgende aanloopblok. In het andere geval worden de volgende zes karakters ingelezen (72D4H) en in FILNM2 gezet.

Indien FILNAM enkel spaties bevat, wordt niet gecontroleerd of de gevonden file de gevraagde naam draagt, en wordt er van uit gegaan dat het goede identificatieblok werd gevonden. In het andere geval wordt de inhoud van FILNAM vergeleken met die van FILNM2, om te zien of de gevonden file wel de gevraagde is. Komen de namen niet overeen, en werkt de Interpreter in directe mode, dan wordt de tekst "Skip :" afgedrukt (710DH), gevolgd door de gevonden naam. Daarna keert de routine terug naar het beginpunt, om het volgende aanloopblok te lezen. Komen beide namen wel overeen, en de Interpreter werkt in directe mode, dan wordt de tekst "Found:" afgedrukt (710DH), gevolgd door de filenaam, en de routine stopt.

adres 70FFH

Hier staat enkel de tekst "Found:", gevolgd door een byte nul.

adres 7106H

Hier staat enkel de tekst "Skip :", gevolgd door een byte nul.

adres 710DH

Tenzij CURLIN aangeeft dat de Interpreter een programma verwerkt, drukt deze routine de tekst af (6678H) waarvan het startadres in registerpaar HL staat, gevolgd door de zes karakters die in FILNM2 staan.

adres 7125H

Deze routine schrijft een identificatieblok op een cassette. Ze wordt gebruikt door de "CSAVE" en "BSAVE"-routines, en maakt ook een deel uit van de open-functie van de routine die de In/Out-bewerkingen afhandelt (6F8FH) indien het apparaat "CAS" is en in output-mode wordt gewerkt. Bij de start staat de naam van de file in FILNAM. Register A bevat het type van de file : D3H voor een gecodeerde BASIC-file (CSAVE), D0H voor een binaire file (BSAVE) en EAH voor een ASCII-file (SAVE of datafile).

De motor van de cassetterecorder wordt aangezet en er wordt een lang aanloopblok naar de cassette geschreven (72F8H). Daarna wordt het type-byte naar cassette geschreven (72DEH), gevolgd door tien keer de eerste zes karakters in FILNAM (72DEH). Dan wordt de motor afgezet via de standaardroutine TAPOOF, en de routine stopt.

adres 713EH

Deze routine wordt door het "CSAVE"-statement gebruikt om het programmageheugen als één blok op cassette te zetten. Alle adrespointers in het programmageheugen worden terug in regelnummers omgezet (54EAH) om het programma onafhankelijk te

maken van het startadres. De motor van de cassetterecorder wordt aangezet en er wordt een kort aanloopblok naar de cassette gestuurd (72F8H). Dan wordt, één byte tegelijk, het hele programmageheugen naar de cassette geschreven (72DEH), gevolgd door zeven nul-bytes als eind-markeerder (72DEH). Tenslotte wordt de motor van de cassetterecorder afgezet via de standaardroutine TAPOOF, en de routine stopt.

adres 715DH

Deze routine wordt gebruikt door de "CLOAD" en "CLOAD?"-routines om één blok gegevens in het programmageheugen te laden of het met de huidige inhoud te vergelijken. Bij de start van de routine bevat register A een code die het onderscheid aangeeft tussen de twee genoemde statements : 00H is "CLOAD" en FFH is "CLOAD?". De motor van de cassetterecorder wordt aangezet en het eerste aanloopblok wordt gezocht (72E9H). Een reeks karakters worden nu van de cassette gelezen (72D4H) en in het geheugen gezet ofwel vergeleken met de huidige inhoud ervan. Gaat het om een "CLOAD?"-statement, dan stopt de routine met vlag NZ indien het ingelezen karakter verschilt van het overeenkomstig karakter in het geheugen. In het andere geval worden de gegevens ingelezen tot een reeks van tien nul-bytes wordt gevonden. Die groep bestaat uit de eind-markeerder van de laatste programmaregel, het laatste schakeladres en de zeven nullen die de "CSAVE"-routine had geschreven. Merk op dat de routine vermoedelijk tijdens het inlezen van die groep zal stoppen, indien ze door "CLOAD?" wordt gebruikt, omdat dan nog steeds met de inhoud van het geheugen wordt vergeleken. Dit verklaart de nogal eigenaardige manier waarop het laatste stuk code van de "CLOAD?"-routine geschreven werd.

adres 71B2H TABEL

Deze tabel wordt door de dispatcher gebruikt voor het decoderen van functiecodes in verband met "GRP". Hij bevat het adres van de routines die alle functiecodes verwerken. De meeste ervan zijn eigenlijk foutmeldings-codes.

ADRES	FUNCTIE
71B6H	0, file openen
71C2H	2, file sluiten
6E86H	4, random files
7196H	6, sequentiële output
475AH	8, sequentiële input
475AH	10, loc
475AH	12, lof
475AH	14, eof
475AH	16, fpos
475AH	18, terugzetten

adres 7196H

Deze routine wordt door de dispatcher gebruikt voor sequentiële output naar het "GRP"-scherm. Via SCRMOD wordt gecontroleerd of het scherm in een van beide tekst-modes staat, en zo ja wordt de foutmelding "Illegal function call" gegeven (475AH). In het andere geval wordt het af te drukken karakter uit register C

gelezen en de controle gaat naar de standaardroutine GRPPRT.

adres 71A2H TABEL

Deze tabel wordt door de dispatcher gebruikt voor het decoderen van functiecodes in verband met "CRT". Hij bevat het adres van de routines die alle functiecodes verwerken. De meeste ervan zijn eigenlijk foutmeldings-codes.

ADRES	FUNCTIE
71B6H	0, file openen
71C2H	2, file sluiten
6E86H	4, random files
71C3H	6, sequentiële output
475AH	8, sequentiële input
475AH	10, loc
475AH	12, lof
475AH	14, eof
475AH	16, fpos
475AH	18, terugzetten

adres 71B6H

Deze routine wordt door de dispatcher gebruikt om een kanaal te openen naar CRT, LPT en GRP. De gevraagde mode, die in register E staat, wordt gecontroleerd en indien blijkt dat "input" of "append" wordt gevraagd, wordt de foutmelding "Bad file name" gegeven (6E6BH). Het adres van het controleblok wordt in PTRFIL gezet, de mode in byte 0 van het controleblok en de routine stopt. De RET-instructie voor de Z80 op het einde van deze routine (71C2H) dient als "CLOSE"-routine voor de dispatcher, wanneer het gaat om CRT, LPT of GRP.

adres 71C3H

Deze routine wordt door de dispatcher gebruikt voor sequentiële output naar de CRT. Het af te drukken karakter wordt uit register C gelezen en de controle gaat over naar de standaardroutine CHPUT.

adres 71C7H TABEL

Deze tabel wordt door de dispatcher gebruikt voor het decoderen van functiecodes in verband met "CAS". Hij bevat het adres van de routines die alle functiecodes verwerken. Vele ervan zijn eigenlijk foutmeldings-codes.

ADRES	FUNCTIE
71DBH	0, file openen
7205H	2, file sluiten
6E86H	4, random files
722AH	6, sequentiële output
723FH	8, sequentiële input
475AH	10, loc
475AH	12, lof
726DH	14, eof
475AH	16, fpos
727CH	18, terugzetten

adres 71DBH

Deze routine wordt door de dispatcher gebruikt om een kanaal te openen naar "CAS". De huidige positie in de In/Out-buffer (die in byte 6 van het controleblok staat) en CASPRV, waarin de eventuele teruggezette karakters staan, worden op nul gezet. In register E staat de gevraagde mode. Indien die "random" of "append" blijkt te zijn, wordt de foutmelding "Bad file name" gegeven (6E6BH). Gaat het om output, dan wordt het identificatieblok naar de cassette gestuurd (7125H). Gaat het om input, dan wordt het correcte identificatieblok op de cassette gelocaliseerd (70B8H). Daarna wordt het adres van het controleblok in PTRFIL gezet, de mode in byte Ø van het controleblok, en de routine stopt.

adres 7205H

Deze routine wordt door de dispatcher gebruikt om het "CAS"-kanaal te sluiten. Byte Ø van het controleblok wordt onderzocht en indien dit input aangeeft, wordt CASPRV nul gemaakt en de routine stopt. In het andere geval wordt de rest van de In/Out-buffer opgevuld met "end of file"-karakters (1AH) en de inhoud van de In/Out-buffer wordt naar de cassette geschreven (722FH). Tenslotte wordt CASPRV nul gemaakt, en de routine stopt.

adres 722AH

Deze routine gebruikt de dispatcher voor sequentiële output naar "CAS". Het te sturen karakter wordt uit register C gelezen en op de eerste vrije plaats in de In/Out-buffer gezet (728BH). Byte 6 van het controleblok, dat de positie in de buffer bevat, wordt opgehoogd. Blijkt dat de bufferpositie doortelt naar nul, dan wil dat zeggen dat de In/Out-buffer tweehonderd zesenvijftig karakters bevat, en dat de inhoud van de buffer naar cassette geschreven moet worden. De motor van de cassetterecorder wordt aangezet en een kort aanloopblok wordt naar cassette geschreven (72F8H), gevolgd door de inhoud van de In/Out-buffer (72DEH). Daarna wordt, via de standaardroutine TAPOOF, de motor afgezet.

adres 723FH

Deze routine wordt door de dispatcher gebruikt voor sequentiële input vanuit "CAS". Eerst wordt in CASPRV gecontroleerd (72BEH) of er een teruggezet karakter in staat; in dat geval is zijn inhoud niet nul. Is dat het geval, dan wordt dit karakter in register A geladen. Zoniet, wordt aan de hand van de positie in de In/Out-buffer gecontroleerd (729BH) of daar karakters staan. Is de In/Out-buffer leeg, dan wordt de motor van de cassetterecorder aangezet en een aanloopblok gelocaliseerd (72E9H). Daarna worden tweehonderd zesenvijftig karakters ingelezen (72D4H), de motor van de cassetterecorder via de standaardroutine TAPION afgezet, en de positie in de In/Out-buffer op nul gezet. Vervolgens wordt het karakter op de huidige positie in de In/Out-buffer opgehaald, en de positie wordt opgehoogd. Tenslotte wordt gecontroleerd of het 'n "end of file"-karakter is (1AH). Is dat niet zo, dan stopt de routine met het karakter in register A en vlag NC. Was het wel het geval, dan wordt het karakter in CASPRV gezet, zodanig dat een latere vraag om input telkens een "end of file"-status zal

opleveren. Hier stopt de routine met vlag C.

adres 726DH

Deze routine gebruikt de dispatcher als de "EOF"-routine voor "CAS". Het volgende karakter wordt ingelezen (723FH) en in CASPRV gezet. Het wordt vergeleken met de "end of file"-code (1AH) en het resultaat van die vergelijking wordt als een geheel getal in DAC gezet, nul indien ze ongelijk zijn, FFFFH indien ze gelijk zijn.

adres 727CH

Deze routine gebruikt de dispatcher als terugzetroutine voor "CAS". Het karakter wordt gewoon in CASPRV gezet, waar het zal opgehaald worden bij de eerstvolgende vraag om sequentiële input.

adres 7281H

Deze routine wordt door de "CLOSE"-functie van de dispatcher aangeroepen, om te controleren of er karakters in de In/Out-buffer staan, en daarna het positie-byte in het controleblok van die buffer nul te maken.

adres 728BH

Deze routine wordt door de sequentiële output-functie van de dispatcher gebruikt om het karakter dat in register A staat, op de huidige positie in de In/Out-buffer te zetten. Daarna wordt de positie opgehoogd.

adres 729BH

Deze routine wordt door de sequentiële input-functie van de dispatcher gebruikt om het karakter te lezen dat staat op de huidige positie in de In/Out-buffer. De positie wordt vervolgens opgehoogd.

adres 72A6H TABEL

Deze tabel wordt door de dispatcher gebruikt tijdens het decoderen van functiecodes voor "LPT". Hij bevat de adressen van de routines die alle functiecodes verwerken. De meeste ervan zijn eigenlijk foutmeldings-codes.

ADRES	FUNCTIE
71B6H	0, file openen
71C2H	2, file sluiten
6E86H	4, random files
72BAH	6, sequentiele output
475AH	8, sequentiele input
475AH	10, loc
475AH	12, lof
475AH	14, eof
475AH	16, fpos
475AH	18, terugzetten

adres 72BAH

Deze routine wordt door de dispatcher gebruikt voor sequentiële output naar "LPT". Het te sturen karakter wordt uit register C gelezen en de controle gaat over naar de standaardroutine OUTDLP.

adres 72BEH

Deze routine wordt door de sequentiële input-functie van de dispatcher gebruikt om te controleren of in CASPRV een teruggezet karakter staat en vlag Z te setten indien dat zo is. In het andere geval wordt CASPRV nul gemaakt, en het karakter wordt vergeleken met het "end of file"-karakter (1AH). Is het een ander karakter, dan wordt dat in register A geladen en de vlaggen worden op NZ, NC gezet. Is het een "end of file"-karakter, dan wordt het terug in CASPRV gezet en de vlaggen worden op Z, C gezet.

adres 72CDH

Deze routine wordt door diverse functies van de dispatcher gebruikt om te controleren of de mode in register E, "append" is, en om, indien dit zo is, de foutmelding "Bad file name" te produceren (6E6BH).

adres 72D4H

Deze routine wordt door diverse functies van de dispatcher gebruikt om een karakter van de cassette in te lezen. Het karakter wordt gelezen via de standaardroutine TAPIN. Indien bij terugkeer vlag C geset staat, wordt de foutmelding "Device I/O error" gegeven (73B2H).

adres 72DEH

Deze routine wordt door diverse functies van de dispatcher gebruikt om een karakter naar de cassette te sturen. Het karakter wordt geschreven via de standaardroutine TAPOUT. Indien bij terugkeer vlag C geset staat, wordt de foutmelding "Device I/O error" gegeven (73B2H).

adres 72E9H

Deze routine wordt door diverse functies van de dispatcher gebruikt om de motor van de cassetterecorder aan te zetten, voor input vanaf de cassette. De motor wordt aangezet via de standaardroutine TAPION. Indien bij terugkeer vlag C geset staat, wordt de foutmelding "Device I/O error" gegeven.

adres 72F8H

Deze routine wordt door diverse functies van de dispatcher gebruikt om de motor van de cassetterecorder aan te zetten, voor output naar de cassette. De controle wordt gewoon overgedragen aan de standaardroutine TAPOON.

adres 7304H

Deze routine wordt gebruikt door het "Ok"-punt van de Hoofdlus

van de Interpreter, door de "END"-routine en door de "RUN-CLEAR"-routine, om de printer te stoppen. Eerst wordt PRTFLG nul gemaakt, en dan wordt in LPTPOS gecontroleerd of er nog karakters naar de printer werden gestuurd, die in de lijnbuffer van de printer blijven wachten. Zo ja, wordt een "CR"/"LF"-combinatie naar de printer gestuurd, om de buffer leeg te schrijven, en LPTPOS wordt nul gemaakt.

adres 7323H

Deze routine stuurt, via de standaardroutine OUTDO, een "CR"/"LF"-combinatie naar het apparaat dat op dat ogenblik output verzorgt. Daarna wordt LPTPOS of TTYPOS nul gemaakt, afhankelijk van het feit of de printer dan wel het scherm wordt gebruikt voor output.

adres 7347H

Deze routine wordt door de Factor Evaluator gebruikt om de "INKEY$"-functie toe te passen. De toestand van de toetsenbord-buffer wordt onderzocht via de standaardroutine CHSNS. Is de buffer leeg, dan wordt het adres van een loze lege string in DAC gezet. Is de buffer niet leeg, dan wordt het eerstvolgende karakter uit de buffer gelezen, via de standaardroutine CHGET. Na een controle of er voldoende ruimte is (6625H) wordt het karakter in het stringgeheugen gezet en een signalement opgesteld (6821H).

adres 7367H

Deze routine wordt door de "LIST"-routine gebruikt om, via de standaardroutine OUTDO, een karakter naar het actieve output-apparaat te sturen. Is het karakter een "LF"-code, dan wordt eveneens een "CR"-code gestuurd.

adres 7374H

Deze routine wordt door de Hoofdlus van de Interpreter gebruikt om een regel tekst in te lezen, indien de input loopt via een In/Out-buffer in plaats van het toetsenbord, dat wil dus zeggen wanneer een "LOAD"-statement wordt uitgevoerd. De karakters worden sequentieel ingelezen (6C71H) en in BUF gezet totdat ofwel BUF vol is, ofwel een "CR"-code gelezen wordt, ofwel het einde van het in te lezen blok wordt bereikt. Alle karakters worden geaccepteerd, behalve "LF"-codes, die weggelaten worden. Indien BUF vol is of een "CR"-code wordt gelezen, brengt de routine de regel naar de Hoofdlus. Wordt de eindmarkeerder gelezen, en staan er nog een aantal karakters in BUF, dan wordt de regel naar de Hoofdlus gebracht. Wordt de eindmarkeerder gelezen, en is BUF leeg, dan wordt In/Out-buffer nummer 0 gesloten (6D7BH) en in FILNAM gecontroleerd of het programma automatisch moet starten. Moet dat niet, dan wordt de controle overgedragen naar het "Ok"-punt van de Interpreter (411EH). In het andere geval wordt het hele systeem op de inschakel-positie gezet (629AH) en de controle wordt aan de Verwerkingslus overgegeven (4601H), die het programma zal uitvoeren.

adres 73B2H

Hier wordt de foutmelding "Device I/O error" geproduceerd.

adres 73B7H

Deze routine verwerkt het "MOTOR"-statement. Volgt er geen operand, dan wordt de controle overgedragen aan de standaardroutine STMOTR, met in register A de waarde FFH. Volgt na het statement het "OFF"-token (EBH), dan gaat de controle over met de waarde 00H in register A. Volgt na het statement het "ON"-token (95H), dan gaat de controle over met de waarde 01H in register A.

adres 73CAH

Deze routine verwerkt het "SOUND"-statement. De operand met het registernummer, dat lager moet zijn dan veertien, wordt geëvalueerd (521CH) en in register A geladen. De operand met gegevens wordt dan geëvalueerd (521CH) en bit 7 wordt geset, bit 6 gereset, om te vermijden dat de werking van de hulp-poorten van de PSG wordt gewijzigd. De laatste operand wordt in register E geladen en de controle wordt overgegeven aan de standaardroutine WRTPSG.

adres 73E4H

Hier staat één ASCII-spatie. Die wordt door het "PLAY"-statement gebruikt om een operand die een lege string aangeeft, te vervangen door een blanco string met één karakter.

adres 73E5H

Deze routine verwerkt het "PLAY"-statement. Het adres van de tabel met commando's voor het statement, op adres 752EH, wordt in MCLTAB gezet, ten behoeve van de macro-ontleder, en PRSCNT wordt nul gemaakt. De eerste string-operand, die verplicht aanwezig is, wordt geëvalueerd (4C64H), zijn opslagruimte vrijgemaakt (67D0H) en zijn lengte en adres in VCBA gezet, in bytes 2, 3 en 4. De stackpointer van het kanaal wordt op VCBA+33 geïnitialiseerd en in VCBA gezet, in bytes 5 en 6. Volgt er in het statement nog meer tekst, dan wordt dit hele proces herhaald voor de kanalen B en C, tot maximaal drie operands geëvalueerd werden. Volgen er nog meer, dan wordt de foutmelding "Syntax error" gegeven (4055H). Bevat het statement minder dan drie strings, dan wordt een eindmarkeerder (FFH) in de muziekrij van elk ongebruikt kanaal gezet (7507H). Register A wordt nul gemaakt, waardoor kanaal A wordt geselecteerd, en de routine loopt door in de hoofdlus van het muziekgedeelte.

adres 744DH

Hier start de hoofdlus van het muziekgedeelte. Het aantal vrije bytes in de huidige muziekrij wordt berekend (7521H) en indien dat minder dan acht is, wordt het volgende kanaal geselecteerd (74D6H) om niet te hoeven wachten tot de rij afgewerkt is. Het stuk dat nog overblijft van de string-operand wordt nu uit de huidige kanaalbuffer gehaald en indien er nog nul bytes geïnterpreteerd moeten worden, gaat de lus opnieuw verder met het volgende kanaal (74D6H). In het andere geval worden de lengte en het adres van de huidige string uit de kanaalbuffer gehaald en in MCLLEN en MCLPTR geplaatst, ten behoeve van de macro-ontleder. De vorige inhoud van de stack wordt uit de kanaalbuffer naar de stack van de Z80 gecopieerd (6253H), MCLFLG

wordt niet-nul gemaakt en de controle wordt overgedragen aan de macro-ontleder (56A2H).

Normaal tast de macro-ontleder een string karakter na karakter af, met behulp van de routines die de commando's van het "PLAY"-statement verwerken, tot de string afgewerkt is. Indien evenwel een muziekrij vol raakt terwijl een toon wordt gegenereerd, wordt een abnormaal einde geforceerd, terug naar de hoofdlus (748EH), zodat het volgende kanaal verwerkt kan worden zonder te hoeven wachten tot de rij afgewerkt is. Na een normale verwerking, wordt een eindmarkeerder in de huidige rij gezet (7507H) en PRSCNT wordt opgehoogd, om aan te geven hoeveel strings reeds afgewerkt werden. Wordt evenwel de verwerking op een abnormale manier beëindigd, dàn wordt wat er nog op de stack van de Z80 staat, naar de huidige kanaalbuffer gecopieerd (6253H). Omwille van de recursieve werking van de macro-ontleder waar het om het "X"-commando gaat, is het mogelijk dat op het einde nog een aantal string-signalementen van vier bytes op de stack van de Z80 staan, die aangeven waar de oorspronkelijke string onderbroken werd. Door de inhoud van de stack in de kanaalbuffer te plaatsen, kan die weer gebruikt worden op het ogenblik dat de lus opnieuw dat bepaald kanaal aanpakt. Merk op dat de foutmelding "Illegal function call" wordt geproduceerd (475AH) indien er nog teveel gegevens op de stack overblijven. De reden daarvan is, dat er slechts zestien bytes per kanaalbuffer beschikbaar zijn. Die foutmelding treedt op indien een muziekrij vol raakt en er een aantal "geneste" "X"-commando's te verwerken zijn. Een voorbeeld :

```
10 A$="XB$;"
20 B$="XC$;"
30 C$="XD$;"
40 D$=STRING$(150,"A")
50 PLAY A$
```

In dit stukje code lijkt een foutje te zitten : er worden slechts vijftien bytes op de stack toegelaten, in plaats van zestien, voordat een foutmelding wordt gegeven.

Bij terugkeer uit de macro-ontleder wordt register A opgehoogd om het volgende kanaal te selecteren. Zijn alle drie de kanalen verwerkt, dan wordt INTFLG gecontroleerd. Blijkt dat de interrupt-verwerker een "CTRL/STOP" heeft gedetecteerd, dan gaat de controle over naar de standaardroutine GICINI, die de muziek laat ophouden. Indien uit bit 7 van PRSCNT blijkt dat dit de eerste keer is dat de hoofdlus wordt doorlopen, dat wil zeggen dat geen kanaal werd onderbroken omwille van een volle rij, dan wordt PLYCNT opgehoogd en via de standaardroutine STRTMS wordt begonnen met het afhandelen van de rijen door interrupt-routines. Daarna wordt in PRSCNT gecontroleerd hoeveel strings al door de macro-ontleder werden afgewerkt. Zijn de drie string-operands helemaal afgewerkt, dan stopt de routine. In het andere geval gaat de controle weer terug naar het begin van de hoofdlus, om elk kanaal opnieuw te bekijken.

adres 7507H

Deze routine wordt door het "PLAY"-statement gebruikt om een eindmarkeerder (FFH) in de huidige muziekrij te zetten, via de standaardroutine PUTQ. Is de rij vol, dan wacht de routine tot er plaats vrij komt.

adres 7521H

Deze routine wordt door het "PLAY"-statement gebruikt om via de standaardroutine LFTQ te berekenen hoeveel vrije ruimte over is in de huidige muziekrij. Zijn er minder dan acht bytes vrij (de maximale lengte van een muziekpakket is zeven bytes) wordt vlag C geset.

adres 752EH TABEL

De tabel op dit adres bevat de toegelaten commando-letters en de overeenkomstige adressen voor de commando's die bij het "PLAY"-statement horen. De commando's die vergezeld kunnen gaan van een parameter, en waarvan derhalve bit 7 in de tabel geset is, krijgen een asterisk :

COMMANDO	ADRES
A	763EH
B	763EH
C	763EH
D	763EH
E	763EH
F	763EH
G	763EH
M*	759EH
V*	7586H
S*	75BEH
N*	7621H
O*	75EFH
R*	75FCH
T*	75E2H
L*	75C8H
X	5782H

adres 755FH TABEL

Deze tabel wordt gebruikt door de routines die de "A" tot en met "G"-commando's in het "PLAY"-statement verwerken. Met behulp van de tabel wordt een noot-nummer tussen nul en veertien, vertaald in een adres in de tabel op 756EH die delers bevat. Hieronder wordt de noot aangegeven in plaats van het nootnummer, met daarbij de afstand vanaf het startadres van de deler-tabel.

BYTENUMMER in de tabel	NOOT	
16	A-	la ♭
18	A	la
20	A+ of B-	la# of si ♭
22	B of C-	si of do ♭
00	B+	si#
00	C	do
02	C+ of D-	do# of re ♭
04	D	re
06	D+ of E-	re# of mi ♭
08	E of F-	mi of fa ♭
10	E+	mi#
10	F	fa
12	F+ of G-	fa# of sol ♭
14	G	sol
16	G+	sol#

adres 756EH TABEL

Deze tabel bevat de twaalf constanten die de Programmeerbare geluidsgenerator als delers gebruikt, om de tonen van octaaf 1 te produceren. Bij elke constante wordt hieronder de overeenkomstige noot met haar frekwentie gegeven.

DELER	NOOT		FREQUENTIE
3421	C	do	32,698 Hz
3228	C+	do#	34,653 Hz
3047	D	re	36,712 Hz
2876	D+	re#	38,895 Hz
2715	E	mi	41,201 Hz
2562	F	fa	43,662 Hz
2419	F+	fa#	46,243 Hz
2283	G	sol	48,997 Hz
2155	G+	sol#	51,908 Hz
2034	A	la	54,995 Hz
1920	A+	la#	58,261 Hz
1812	B	si	61,773 Hz

adres 7586H

Deze routine verwerkt het "V"-commando van het "PLAY"-statement. De parameter, met een standaardwaarde van acht, wordt in byte 18 van de huidige kanaalbuffer gezet, zonder bit 6 van de bestaande inhoud te wijzigen. Er worden geen muziek-data geproduceerd.

adres 759EH

Deze routine verwerkt het "M"-commando van het "PLAY"-statement. De parameter, met een standaardwaarde van 255, wordt vergeleken met de bestaande modulatieperiode die in bytes 19 en 20 van de huidige kanaalbuffer staat. Zijn ze gelijk, dan stopt de routine zonder meer. Zoniet, wordt de nieuwe modulatieperiode in de kanaalbuffer gezet, en in byte 18 wordt bit 6 geset om aan te geven dat de nieuwe waarde moet worden opgenomen wanneer het eerstvolgende datapakket wordt klaargemaakt. Er worden geen muziek-data geproduceerd.

adres 75BEH

Deze routine verwerkt het "S"-commando van het "PLAY"-statement. De parameter wordt in byte 18 van de huidige kanaalbuffer geplaatst, en bit 4 van datzelfde byte wordt geset, om aan te geven dat de nieuwe waarde moet worden opgenomen wanneer het eerstvolgende datapakket wordt klaargemaakt. Er worden geen muziek-data geproduceerd. De karakteristieken van de Programmeerbare geluidsgenerator maken dat de parameters voor de vorm en het volume elkaar uitsluiten, zodat hetzelfde byte in de kanaalbuffer voor beide gebruikt kan worden.

adres 75C8H

Deze routine verwerkt het "L"-commando van het "PLAY"-statement. De parameter, met een standaardwaarde van vier, wordt in byte 16 van de huidige kanaalbuffer gezet. Daar wordt hij gebruikt bij de berekening van de lengte van opeenvolgende noten. Er worden geen muziek-data geproduceerd.

adres 75E2H

Deze routine verwerkt het "T"-commando van het "PLAY"-statement. De parameter, met een standaardwaarde van 120, wordt in byte 17 van de huidige kanaalbuffer gezet. Daar wordt hij gebruikt bij de berekening van de lengte van opeenvolgende noten. Er worden geen muziek-data geproduceerd.

adres 75EFH

Deze routine verwerkt het "O"-commando van het "PLAY"-statement. De parameter, met een standaardwaarde van vier, wordt in byte 15 van de huidige kanaalbuffer gezet. Daar wordt hij gebruikt bij de berekening van de frekwentie van opeenvolgende noten. Er worden geen muziek-data geproduceerd.

adres 75FCH

Deze routine verwerkt het "R"-commando van het "PLAY"-statement. De lengte-parameter, met een standaardwaarde van vier, wordt in registerpaar DE gezet en een delerwaarde nul in registerpaar HL. De bestaande waarde voor het volume wordt uit byte 18 van de huidige kanaalbuffer gehaald, tijdelijk door een nul vervangen, en de controle wordt overgedragen aan de toongenerator (769CH).

adres 7621H

Deze routine verwerkt het "N"-commando van het "PLAY"-statement. De verplichte parameter wordt eerst onderzocht. Is die nul, dan wordt een rusttijd geproduceerd (760BH). Is hij groter dan zesennegentig, dan wordt de foutmelding "Illegal function call" gegeven (475AH). In het andere geval wordt herhaaldelijk twaalf van de waarde afgetrokken tot aan een "underflow", om een octaafnummer van één tot negen in register E te verkrijgen, en een noot-nummer van nul tot elf in register C. Daarna wordt de controle overgedragen aan de toongenerator (7673H).

adres 763EH

Deze routine verwerkt de commando's "A" tot "G" van het "PLAY"-

statement. De letter die de noot aangeeft, wordt eerst omgezet in een nootnummer tussen 0 en 14. Dit uitgebreide bereik is noodzakelijk, omdat de manier waarop de noten benoemd worden, meebrengt dat er een paar overtollig zijn. Nu wordt de tabel op adres 755FH gebruikt, om de plaats in de tabel met delers te berekenen, en de deler die bij de opgegeven noot hoort, wordt in registerpaar DE geladen. Het octaafnummer wordt uit byte 15 van de huidige kanaalbuffer gelezen, en de deler wordt gehalveerd totdat het correcte octaaf wordt bereikt. Daarna wordt rechtstreeks in de string-operand gekeken (56EEH) of er een parameter volgt die de lengte van de toon aangeeft. Is dat zo, dan wordt hij omgezet (572FH) en in register C geladen. Volgt er geen parameter, dan wordt de standaardwaarde uit byte 16 van de huidige kanaalbuffer gelezen. Vervolgens wordt de tijdsduur van de noot berekend volgens de formule :

tijdsduur (interrupt-eenheden) = 12.000/(lengte*tempo)

Bij een normale waarde voor de lengte (4) en een normaal tempo (120) geeft dit een toon die 25 interrupt-eenheden van elk 20 msec duurt (=500 msec); een halve seconde. Nu wordt in de string-operand gekeken (56EEH) of er "."-karakters volgen en voor elk dergelijk karakter wordt de tijdsduur met anderhalf vermenigvuldigd. Tenslotte wordt de uiteindelijke lengte gecontroleerd en indien blijkt dat die kleiner is dan vijf interrupt-eenheden, wordt ze vervangen door de waarde vijf. Dat wil zeggen dat de kortste noot die op een Europese machine geproduceerd kan worden, één tiende seconde duurt, wat ook het tempo of de opgegeven lengte is.

Nu wordt het muziekpakket van drie, vijf of zeven bytes lang, in bytes 8 tot 14 van de huidige kanaalbuffer samengesteld, voordat het in de muziekrij wordt gezet. De tijdsduur wordt in bytes 8 en 9 van de kanaalbuffer gezet. Het byte dat het volume bevat, en tevens als vlag dient, wordt uit byte 18 gehaald en in byte 10 van de kanaalbuffer gezet. Bit 7 wordt geset waardoor de interrupt-routine die de rijen verwerkt, weet dat het volume moet gewijzigd worden. Is bit 6 van het volume-byte geset, dan wordt de modulatieperiode uit bytes 19 en 20 gelezen en bij het muziekpakket gevoegd in bytes 11 en 12. Verschilt de delerwaarde van nul, dan wordt ook die bij het pakket gevoegd in bytes 11 en 12, als er geen modulatieperiode werd opgegeven, en in bytes 13 en 14 in het andere geval. Tenslotte wordt het aantal bytes in de hoogste drie bits van byte 8 van de kanaalbuffer gezet, waarmee de voorbereiding van het muziekpakket klaar is.

Wanneer de delerwaarde nul is, waarmee een rust wordt aangegeven, dan wordt de inhoud van SAVVOL terug in byte 18 van de statische buffer gezet. Het muziekpakket wordt nu in de huidige muziekrij gezet via de standaardroutine PUTQ, en er wordt gecontroleerd hoeveel bytes nog vrij zijn (7521H). Zijn er dat minder dan acht, dan wordt de controle rechtstreeks aan de "PLAY"-routine overgedragen (748EH). In het andere geval gaat de controle normaal over naar de macro-ontleder.

adres 7754H

Hier staat de constante 12.000 (enkele precisie) die wordt gebruikt bij het berekenen van de lengte van een noot.

adres 7758H

Deze routine verwerkt het "PUT"-statement. Register B wordt met 80H geladen en de routine loopt door in de "GET"-routine.

adres 775BH

Deze routine verwerkt het "GET"-statement. Register B wordt nul gemaakt, om het onderscheid te maken tussen "GET" en "PUT". Dan wordt het volgende token in de programmatekst bekeken. Daarna wordt de controle overgedragen aan de "PUT SPRITE"-routine (7AAFH) of naar de Disk BASIC routine voor "GET/PUT" (6C35H).

adres 7766H

Deze routine verwerkt het "LOCATE"-statement. Volgt er een kolom-coördinaat, dan wordt hij geëvalueerd (521CH) en in register D geladen. Werd hij niet opgegeven, dan wordt het huidige coördinaat uit CSRX gehaald. Volgt een regel-coördinaat dan wordt die geëvalueerd (521CH) en in register E geladen. Werd hij niet opgegeven, dan wordt het huidige regel-coördinaat uit CSRY gehaald. Volgt daarna nog een operand die de cursor wijzigt, dan wordt die geëvalueerd (521CH) en in register A wordt de waarde 78H geladen als de operand nul is (cursor uit) of 79H indien de operand verschilt van nul (cursor aan). Daarna wordt de cursor aan- of uitgezet via de standaardroutine OUTDO, die de reeks ESC, 78H/79H, "5" uitzendt. De coördinaten van de regel en de kolom worden in registerpaar HL geladen, en de cursorpositie wordt ingevuld via de standaardroutine POSIT.

adres 77A5H

Deze routine verwerkt de statements "STOP ON/OFF/STOP". Het adres van het statusbyte in TRPTBL dat bij het randapparaat hoort, wordt in registerpaar HL geladen en de controle gaat over op de "ON/OFF/STOP"-routine (77CFH).

adres 77ABH

Deze routine verwerkt de statements "SPRITE ON/OFF/STOP". Het adres van het statusbyte in TRPTBL dat bij het randapparaat hoort, wordt in registerpaar HL geladen en de controle gaat over op de "ON/OFF/STOP"-routine (77CFH).

adres 77B1H

Deze routine verwerkt de statements "INTERVAL ON/OFF/STOP". Er is geen specifiek "INTERVAL"-token (deze routine wordt aangeroepen indien een "INT"-token werd gevonden). Daarom wordt eerst gecontroleerd of de karakters "E" en "R" en het "VAL"-token (94H) volgen in de programmatekst. Het adres van het byte in TRPTBL dat bij het randapparaat hoort, wordt in registerpaar HL geladen en de controle gaat over op de "ON/OFF/STOP"- routine (77CFH).

adres 77BFH

Deze routine verwerkt de "STRIG ON/OFF"-statements. Het nummer van de vuurknop, tussen nul en vier, wordt geëvalueerd (7C08H) en het adres van het byte in TRPTBL dat bij het randapparaat hoort, wordt in registerpaar HL geladen. Het "ON/OFF/STOP"-token wordt onderzocht en het statusbyte in TRPTBL dienovereenkomstig gewijzigd (77FEH). Daarna gaat de controle rechtstreeks over naar de Verwerkingslus (4612H). Daardoor wordt voorkomen dat binnengekomen interrupts worden verwerkt voor het einde van het volgende statement.

adres 77D4H

Deze routine verwerkt het statement "Key (n) ON/OFF/STOP". Het nummer van de toets, tussen 1 en 10, wordt geëvalueerd (521CH) en het adres van het statusbyte in TRPTBL dat bij de toets hoort, wordt in registerpaar HL geladen. Het "ON/OFF/STOP"-token wordt onderzocht, en het statusbyte in TRPTBL wordt dienovereenkomstig aangepast (77FEH). Bit 0 van het statusbyte in TRPTBL, het "ON-bit", wordt gecopieerd naar de overeenkomstige positie in FNKFLG, waar het gelezen kan worden door de interrupt-routine die het toetsenbord aftast. Daarna gaat de controle rechtstreeks over naar de Verwerkingslus (4612H).

adres 77FEH

Deze routine controleert of één van de tokens die de toestand van de interrupts wijzigt, in de tekst aanwezig is en leidt de controle naar de toepasselijke routine : "ON" (631BH), "OFF" (632BH) of "STOP" (6331H). Staat er geen van die drie tokens, dan wordt de foutmelding "Syntax error" gegeven (4055H).

adres 7810H

Deze routine wordt gebruikt door de "ON DEVICE GOSUB"-routine (490DH) om te controleren of er een token in de tekst staat, dat een bepaald apparaat voorstelt. Indien geen van de tokens in de tekst staat, wordt vlag C geset. In het andere geval wordt het toegangsnummer tot TRPTBL dat bij het randapparaat hoort, in register B geladen, en in register C het maximaal aantal operands die een regelnummer bevat :

DEVICE	TRPTBL#	LIJNNUMMER
KEY	00	10
STOP	10	01
SPRITE	11	01
STRIG	12	05
INTERVAL	17	01

Vervolgens wordt, alléén bij "INTERVAL", de interval-operand geëvalueerd (542FH) en in INTVAL en INTCNT gezet.

adres 785CH

Deze routine wordt door de "ON DEVICE GOSUB"-routine gebruikt (490DH) om het adres van een programmaregel in TRPTBL te zetten. Het toegangsnummer tot TRPTBL, dan in register B moet staan,

wordt met drie vermenigvuldigd en bij het startadres van de tabel opgeteld. Deze bewerking levert het juiste adres in de tabel op. Het adres dat in registerpaar DE staat wordt op de berekende plaats geladen, eerst het LSB en daarna het MSB.

adres 786CH

Deze routine verwerkt het "KEY"-statement. Verschilt het volgende karakter in de programmatekst van het "LIST"-token (93H), dan gaat de controle over op de "KEY n"-routine (78AEH). Was het wel het "LIST"-token, dan wordt elk van de tien strings die bij een functietoets horen, uit FNKSTR gehaald en via de standaardroutine OUTDO afgedrukt, gevolgd door een "CR"/"LF"-combinatie (7328H). Het "DEL"-karakter (7FH) of een controlekarakter lager dan 20H wordt door een spatie vervangen.

adres 78AEH

Deze routine verwerkt de statements "KEY n", "KEY(n) ON/OFF/STOP", "KEY ON" en "KEY OFF". Indien het eerstvolgende karakter in de programmatekst een "(" is, gaat de controle over naar de "KEY(n) ON/OFF/STOP"-routine (77D4H). Is het een "ON"-token (95H), dan gaat de controle naar de standaardroutine DSPFNK. Is het een "OFF"-token (EBH), dan gaat de controle naar de standaardroutine ERAFNK. In de andere gevallen wordt het nummer van de toets geëvalueerd (521CH) en registerpaar DE wordt geladen met het adres in FNKSTR dat bij die toets hoort. De string-operand wordt geëvalueerd (4C64H) en zijn opslagruimte wordt vrijgemaakt (67D0H). Tot maximaal vijftien karakters worden uit de string naar FNKSTR gecopieerd; ongebruikte plaatsen worden opgevuld met nullen. Wordt in de operand een nul-byte gevonden, dan wordt de foutmelding "Illegal function call" gegeven (475AH). Nu wordt de controle overgedragen naar de standaardroutine FNKSB om de tekst van de functietoetsen, indien die op het scherm staat, aan te passen.

adres 7900H

Deze routine wordt door de Factor Evaluator gebruikt om de "TIME"-functie uit te voeren. De inhoud van JIFFY wordt als een getal met enkele precisie in DAC gezet (3236H).

adres 790AH

Deze routine wordt door de Factor Evaluator gebruikt om de "CSRLIN"-functie uit te voeren. De inhoud van CSRY wordt met één verlaagd en als een geheel getal in DAC gezet (2E9AH).

adres 7911H

Deze routine verwerkt het "TIME"-statement. De operand wordt geëvalueerd (542FH) en in JIFFY geladen.

adres 791BH

Deze routine wordt door de Factor Evaluator gebruikt om de "PLAY"-functie uit te voeren. De numerieke operand die het te selecteren kanaal aangeeft, wordt geëvalueerd (7C08H). Is die nul, dan wordt de inhoud van MUSICF als een geheel getal met een waarde 0 of FFFFH in DAC gezet. Is de operand niet nul, dan

wordt met behulp van het kanaalnummer de juiste bit van MUSICF geselecteerd, waarna dit eveneens in een geheel getal wordt omgezet.

adres 7940H

Deze routine wordt door de Factor Evaluator gebruikt om de "STICK"-functie toe te passen op een operand in DAC. Het nummer van de joystick wordt gecontroleerd (521FH) en in register A aan de standaardroutine GTSTCK doorgegeven. Het resultaat wordt als een geheel getal in DAC gezet (4FCFH).

adres 794CH

Deze routine wordt door de Factor Evaluator gebruikt om de "STRIG"-functie toe te passen op een operand in DAC. Het nummer van de vuurknop wordt gecontroleerd (521FH) en in register A aan de standaardroutine GTTRIG doorgegeven. Het resultaat wordt als een geheel getal met de waarde nul of FFFFH in DAC gezet.

adres 795AH

Deze routine wordt door de Factor Evaluator gebruikt om de "PDL"-functie toe te passen op een operand in DAC. Het nummer van de paddle wordt gecontroleerd (521FH) en in register A aan de standaardroutine GTPDL doorgegeven. Het resultaat wordt als een geheel getal in DAC gezet (4FCFH).

adres 7969H

Deze routine wordt door de Factor Evaluator gebruikt om de "PAD"-functie toe te passen op een operand in DAC. Het nummer van de "pad" wordt gecontroleerd (521FH) en in register A aan de standaardroutine GTPAD doorgegeven. Het resultaat wordt in DAC gezet, als een geheel getal voor de "pads" 1, 2, 5 of 6 en als een geheel getal met de waarde nul of FFFFH voor de "pads" 0, 3, 4 of 7.

adres 7980H

Deze routine verwerkt het "COLOR"-statement. Wordt het statement gevolgd door een operand die een voorgrondkleur aangeeft, dan wordt die geëvalueerd (521CH) en in register E geladen, zoniet wordt de huidige voorgrondkleur uit FORCLR gehaald. Wordt het statement gevolgd door een operand die een achtergrondkleur aangeeft, dan wordt die geëvalueerd (521CH) en in register D gezet, zoniet wordt de huidige achtergrondkleur uit BAKCLR gehaald. Wordt het statement gevolgd door een operand die een randkleur aangeeft, dan wordt die geëvalueerd (521CH) en in BDRCLR gezet. De voorgrondkleur wordt in FORCLR en ATRBYT gezet, de achtergrondkleur in BAKCLR. Daarna gaat de controle over naar de standaardroutine CHGCLR, waar de nodige wijzigingen aan de Video Display Processor worden aangebracht.

adres 79CCH

Deze routine verwerkt het "SCREEN"-statement. Volgt op dit statement een mode-operand, dan wordt die geëvalueerd en via register A aan de standaardroutine CHGMOD doorgegeven. Volgt er een operand die een sprite-maat aangeeft, dan wordt die

geëvalueerd (521CH) en in bits 0 en 1 van RG1SAV gezet, de copie van het moderegister 1 van de Video Display Processor, die in het Werkgebied staat. De parameters van de Video Display Processor die op sprites betrekking hebben, worden gewist via de standaardroutine CLRSPR. Volgt er op het statement een operand die aangeeft of er bij het indrukken van een toets al dan niet een klik geproduceerd moet worden, dan wordt die geëvalueerd (521CH) en in CLIKSW gezet : nul voor geen klik, niet-nul voor wél een klik. Volgt er een operand die de baudrate bepaalt, dan wordt die geëvalueerd en de baudrate wordt ingesteld (7A2DH). Volgt er een operand die de printer-mode aangeeft, dan wordt die geëvalueerd (521CH) en in NTMSXP gezet : nul voor een MSX-printer, niet-nul voor een andere printer.

adres 7A2DH

Deze routine wordt gebruikt om de baudrate voor de cassette in te stellen. De operand wordt geëvalueerd (521CH). Daarna worden uit CS1200 of CS2400, al naargelang, vijf bytes naar LOW gecopieerd.

adres 7A48H

Deze routine verwerkt het "SPRITE"-statement. Indien het daaropvolgende karakter geen "$" is, gaat de controle over naar de "SPRITE ON/OFF/STOP"-routine (77ABH). Zoniet wordt in SCRMOD gecontroleerd of het scherm in de 40x24 tekstmode staat. Zo ja, wordt de foutmelding "Illegal function call" gegeven (475AH). Het nummer van het spritepatroon wordt geëvalueerd en zijn plaats in de Patroontabel van de sprites in VRAM wordt opgezocht (7AA0H). Daarna wordt de string-operand geëvalueerd (4C5FH) en zijn opslagruimte vrijgemaakt (67D0H). De grootte van de sprite, die via de standaardroutine GSPSIZ werd berekend, wordt vergeleken met de lengte van de string. Is de string korter dan de sprite, dan wordt de Patroontabel eerst via de standaardroutine FILVRM met nullen gevuld. Vervolgens worden via de standaardroutine LDIRVM karakters uit de string gecopieerd naar de Patroontabel totdat ofwel het einde van de string bereikt is, ofwel de sprite vol is. Indien de string langer is dan de grootte van de sprite, worden de overtollige karakters genegeerd.

adres 7A84H

Deze routine wordt door de Factor Evaluator gebruikt om de "SPRITE$"-functie toe te passen. Het nummer van het spritepatron wordt geëvalueerd en de plaats ervan in de Patroontabel van de sprites in VRAM wordt berekend (7A9FH). De grootte van de sprite, die via de standaardroutine GSPSIZ werd berekend, wordt nu in registerpaar BC geladen om het aantal te copiëren bytes te tellen. Na een controle of er voldoende ruimte vrij is in het stringgeheugen (6627H) wordt het spritepatroon via de standaardroutine LDIRMV uit VRAM gecopieerd en het signalement van de verkregen string wordt opgesteld (6654H). Doordat de schermmode niet wordt gecontroleerd tijdens deze functie, kunnen enkele interessante neveneffecten bereikt worden. Dit wordt verderop in dit boek behandeld.

adres 7A9FH

Deze routine wordt door het "SPRITE$"-statement en de "SPRITE$"-functie gebruikt om een spritepatroon op te zoeken in de Patroontabel in VRAM. Het patroonnummer wordt geëvalueerd (7C08H) en in register A aan de standaardroutine CALPAT doorgegeven. Het adres van het patroon wordt in registerpaar DE geladen en de routine stopt.

Merk op dat de grootte van het patroonnummer bij verschillende sprite-groottes, niet wordt gecontroleerd. Patroonnummers tot en met 255 worden geaccepteerd, zelfs voor 16x16-sprites, terwijl de maximale grootte 63 is. Dit heeft tot gevolg dat VRAM-adressen boven 3FFFH worden berekend, die dus doortellen naar het lagere VRAM-gebied. Daardoor zal het "SPRITE$"-statement de Karaktertabel in de war brengen. Een voorbeeld :

```
10 SCREEN 3,2
20 SPRITE$(0)=STRING$(32,255)
30 PUT SPRITE 0,(0,0),,0
40 SPRITE$(65)=STRING$(32,255)
50 GOTO 50
```

Het bovenstaande programma plaatst een echte sprite in de linker bovenhoek van het scherm en verknoeit vervolgens met het statement op regel 40 de VRAM rechts ervan. Ook de "SPRITE$"-functie kan op die manier worden gemanipuleerd : aangezien er geen controle op de schermmode plaatsvindt, kunnen tot 32 bytes uit de Namentabel worden gelezen, in de 40x24 tekstmode, bijvoorbeeld zo :

```
10 SCREEN 0,2
20 PRINT "een tekst"
30 A$=SPRITE$(64)
40 PRINT A$
```

adres 7AAFH

Deze routine verwerkt de statements "GET/PUT SPRITE". De controle wordt naar deze routine overgedragen door de algemene "GET/PUT"-routine (775BH). Eerst wordt in register B gecontroleerd of het statement wel degelijk "PUT" is. Klopt dat niet, dan wordt de foutmelding "Illegal function call" gegeven (475AH). In SCRMOD wordt gecontroleerd of het scherm in 40x24 tekstmode staat en indien dit zo is, wordt de foutmelding "Illegal function call" gegeven (475AH). De operand met het spritenummer, tussen 0 en 31, wordt geëvalueerd (521CH) en aan de standaardroutine CALATR doorgegeven, waar het vier bytes lange attributenblok in de Attributentabel wordt gelocaliseerd. Volgt er een operand voor de coördinaten, dan wordt die geëvalueerd en het X-coördinaat wordt in registerpaar BC geladen, het Y-coördinaat in registerpaar DE (579CH).

Het LSB van het Y-coördinaat wordt, via de standaardroutine WRTVRM, in byte 0 van het attributenblok in VRAM gezet. Daarna

wordt bit 7 van het X-coördinaat gecontroleerd, om te bepalen of het negatief is, dat wil zeggen buiten de linker schermrand. Is dat zo, dan wordt er 32 bij opgeteld en register B wordt met 80H geladen, waardoor de "early clock"-bit in het attributenblok geset wordt. Een X-coördinaat -1 (FFFFH) zou bijvoorbeeld gewijzigd worden in +31 met een "early clock". Het LSB van het X-coördinaat wordt dan, via de standaardroutine WRTVRM, in byte 1 van het attributenblok gezet. Via de standaardroutine RDVRM wordt byte 3 van dat blok ingelezen, versmolten met het nieuwe "early clock"-bit, en het wordt via de standaardroutine WRTVRM terug in VRAM gezet.

Indien een kleur-operand volgt, wordt hij geëvalueerd (521CH). Byte 3 van het attributenblok wordt via de standaardroutine RDVRM ingelezen, en de nieuwe kleurcode wordt in de laagste vier bits gezet, waarna de standaardroutine WRTVRM het byte terug in VRAM zet.Volgt een patroonnummer, dan wordt dit geëvalueerd (521CH). De grootte ervan wordt in verband gebracht met de huidige grootte van de sprites, zoals die door de standaardroutine GSPSIZ werd berekend. Het hoogste patroonnummer voor 8x8 sprites is 255, voor 16x16 sprites is dat 63. Het patroonnummer wordt door de standaardroutine WRTVRM in byte 2 van het attributenblok gezet en de routine stopt.

adres 7B37H

Deze routine verwerkt het statement "VDP". De operand met het registernummer, van nul tot zeven, wordt geëvalueerd (7C08H), gevolgd door de operand die het gegeven bevat (521CH). Het registernummer wordt in register C geladen, het databyte in register B en de controle wordt overgedragen aan de standaardroutine WRTVDP.

adres 7B47H

Deze routine wordt door de Factor Evaluator gebruikt om de "VDP"-functie toe te passen. De operand met het registernummer, van nul tot zeven, wordt geëvalueerd (7C08H) en opgeteld bij RG0SAV, om het overeenkomstige registermasker in het Werkgebied te vinden. Het registermasker wordt gelezen en als een geheel getal in DAC gezet (4FCFH).

adres 7B5AH

Deze routine verwerkt het "BASE"-statement. De operand die het nummer van de tabel in de Video Display Processor aangeeft, tussen nul en negentien, wordt geëvalueerd (7C08H), gevolgd door de operand die het base-adres aangeeft (4C64H). Na een controle of het base-adres lager is dan 4000H (7BFEH) wordt met behulp van het tabelnummer de bijbehorende plaats in de maskertabel op adres 7BA3H opgezocht. Het base-adres wordt geAND met het gevonden masker en indien blijkt dat bits op niet toegestane posities geset zijn, wordt de foutmelding "Illegal function call" gegeven (475AH). Het nummer van de tabel in de Video Display Processor wordt opgeteld bij TXTNAM, om het huidige base-adres in het Werkgebied te localiseren, en op die plaats wordt het nieuwe base-adres gezet. Het nummer van de tabel in de Video Display Processor wordt door vijf gedeeld, om te bepalen bij welke van de vier schermmodes de tabel hoort. Is die mode dezelfde als de huidige schermmode, dan wordt het nieuwe base-adres ook naar de Video Display Processor geschreven (7B99H).

adres 7B99H

Deze routine wordt door de "BASE"-routine gebruikt om de base-adressen van de Video Display Processor aan te passen. De huidige schermmode, die in register A staat, wordt onderzocht en de controle wordt overgedragen naar de standaardroutine SETTXT, SETT32, SETGRP of SETMLT, al naargelang. Merk op dat de Video Display Processor niet kompleet wordt geïnitialiseerd, en dat de huidige adressen van de tabellen (NAMBAS, CGPBAS, PATBAS en ATRBAS), die eigenlijk door de schermroutines worden gebruikt, niet aangepast worden. Dit kan aangetoond worden door het volgende programma, waarin de Interpreter output blijft sturen naar de oude tabel in VRAM :

```
10 SCREEN 0
20 BASE(0)=&H400
30 PRINT "een tekst"
40 FOR N=1 TO 2000: NEXT
50 BASE (0)=0
```

Merk ook op dat deze routine een bug bevat. Terwijl SETTXT terecht voor de 40x24 tekstmode wordt gebruikt, wordt SETGRP gebruikt voor de 32x24 tekstmode en SETMLT voor de grafische en de veelkleurenmode. Daarom moet na elk "BASE"-statement een "SCREEN"-statement volgen, om een komplete initialisatie uit te voeren.

adres 7BA3H TABEL

Deze maskertabel wordt door de "BASE"-routine gebruikt om ervoor te zorgen dat enkel toegestane base-adressen voor de Video Display Processor worden geaccepteerd. Naast elk masker wordt het tabelnummer en de overeenkomstig variabele in het Werkgebied aangegeven :

MASKER	TABEL#	SYSTEEMVARIABELE
03FFH	00	TXTNAM
003FH	01	TXTCOL
07FFH	02	TXTCGP
007FH	03	TXTATR
07FFH	04	TXTPAT
03FFH	05	T32NAM
003FH	06	T32COL
07FFH	07	T32CGP
007FH	08	T32ATR
07FFH	09	T32PAT
03FFH	10	GRPNAM
1FFFH	11	GRPCOL
1FFFH	12	GRPCGP
007FH	13	GRPATR
07FFH	14	GRPPAT
03FFH	15	MLTNAM
003FH	16	MLTCOL
07FFH	17	MLTCGP
007FH	18	MLTATR
07FFH	19	MLTPAT

adres 7BCBH

Deze routine wordt door de Factor Evaluator gebruikt om de "BASE"-functie toe te passen. De operand die het tabelnummer aangeeft, van nul tot negentien, wordt geëvalueerd (7C08H) en bij TXTNAM opgeteld, om het gewenste base-adres in het Werkgebied te localiseren. Dit wordt vervolgens als een getal met enkele precisie in DAC gezet (3236H).

adres 7BE2H

Deze routine verwerkt het "VPOKE"-statement. De operand die het adres in VRAM aangeeft, wordt geëvalueerd, en gecontroleerd of hij lager is dan 4000H (7BFEH). Daarna wordt de data-operand geëvalueerd (521CH) en in register A aan de standaardroutine WRTVRM doorgegeven, die hem naar het opgegeven adres schrijft.

adres 7BF5H

Deze routine wordt door de Factor Evaluator gebruikt om de "VPEEK"-functie toe te passen op een operand in DAC. Er wordt gecontroleerd of de operand die het adres in VRAM aangeeft, kleiner is dan 4000H (7BFEH). De standaardroutine RDVRM leest dan het byte uit VRAM en het resultaat wordt als een geheel getal in DAC gezet (4FCFH).

adres 7BFEH

Deze routine zet een numerieke operand die in DAC staat, om in een geheel getal (2F8AH) en laadt dit in registerpaar HL. Is de operand groter dan of gelijk aan 4000H, waardoor hij buiten het toegestane VRAM-bereik ligt, wordt de foutmelding "Illegal function call" gegeven (475AH).

adres 7C08H

Deze routine evalueert (521CH) een numerieke operand tussen haakjes, en zet die als een geheel getal in register A. Is hij groter dan de maximaal toegestane waarde die oorspronkelijk in register A werd aangebracht, dan wordt de foutmelding "Illegal function call" gegeven (475AH).

adres 7C16H

Deze routine verwerkt het statement "DSKO$". Op een standaard MSX-machine levert het de foutmelding "Illegal function call" op (475AH).

adres 7C1BH

Deze routine verwerkt het statement "SET". Op een standaard MSX-machine levert het de foutmelding "Illegal function call" op (475AH).

adres 7C20H

Deze routine verwerkt het statement "NAME". Op een standaard MSX-machine levert het de foutmelding "Illegal function call" op (475AH).

adres 7C25H

Deze routine verwerkt het statement "KILL". Op een standaard MSX-machine levert het de foutmelding "Illegal function call" op (475AH).

adres 7C2AH

Deze routine verwerkt het statement "IPL". Op een standaard MSX-machine levert het de foutmelding "Illegal function call" op (475AH).

adres 7C2FH

Deze routine verwerkt het statement "COPY". Op een standaard MSX-machine levert het de foutmelding "Illegal function call" op (475AH).

adres 7C34H

Deze routine verwerkt het statement "CMD". Op een standaard MSX-machine levert het de foutmelding "Illegal function call" op (475AH).

adres 7C39H

Deze routine wordt door' de Factor Evaluator gebruikt om de "DSKF"-functie toe te passen op een operand in DAC. Op een standaard MSX-machine levert ze de foutmelding "Illegal function call" op (475AH).

adres 7C3EH

Deze routine wordt door de Factor Evaluator gebruikt om de "DSKI$"-functie uit te voeren. Op een standaard MSX-machine levert ze de foutmelding "Illegal function call" op (475AH).

adres 7C43H

Deze routine wordt door de Factor Evaluator gebruikt om de "ATTR$"-functie uit te voeren. Op een standaard MSX-machine levert ze de foutmelding "Illegal function call" op (475AH).

adres 7C48H

Deze routine verwerkt het statement "LSET". Op een standaard MSX-machine levert het de foutmelding "Illegal function call" op (475AH).

adres 7C4DH

Deze routine verwerkt het statement "RSET". Op een standaard MSX-machine levert het de foutmelding "Illegal function call" op (475AH).

adres 7C52H

Deze routine verwerkt het statement "FIELD". Op een standaard MSX-machine levert het de foutmelding "Illegal function call" op

(475AH).

adres 7C57H

Deze routine wordt door de Factor Evaluator gebruikt om de "MKI$"-functie toe te passen op een operand in DAC. Op een standaard MSX-machine levert ze de foutmelding "Illegal function call" op (475AH).

adres 7C5CH

Deze routine wordt door de Factor Evaluator gebruikt om de "MKS$"-functie toe te passen op een operand in DAC. Op een standaard MSX-machine levert ze de foutmelding "Illegal function call" op (475AH).

adres 7C61H

Deze routine wordt door de Factor Evaluator gebruikt om de "MKD$"-functie toe te passen op een operand in DAC. Op een standaard MSX-machine levert ze de foutmelding "Illegal function call" op (475AH).

adres 7C66H

Deze routine wordt door de Factor Evaluator gebruikt om de "CVI"-functie toe te passen op een operand in DAC. Op een standaard MSX-machine levert ze de foutmelding "Illegal function call" op (475AH).

adres 7C6BH

Deze routine wordt door de Factor Evaluator gebruikt om de "CVS"-functie toe te passen op een operand in DAC. Op een standaard MSX-machine levert ze de foutmelding "Illegal function call" op (475AH).

adres 7C70H

Deze routine wordt door de Factor Evaluator gebruikt om de "CVD"-functie toe te passen op een operand in DAC. Op een standaard MSX-machine levert ze de foutmelding "Illegal function call" op (475AH).

adres 7C76H

Deze routine verzorgt de afwerking van de initialisatie bij het inschakelen van de machine. Op dit punt is het hele Werkgebied gevuld met nullen en zijn enkel EXPTBL en SLTTBL geïnitialiseerd. Op adres F376H wordt een tijdelijke stack gemaakt, en alle honderdtwaalf hooks (560 bytes) worden met RET-opcodes voor de Z 80 gevuld (C9H). HIMEM wordt op F380H gezet en het laagste RAM-adres dat werd gevonden (7D5DH), wordt in BOTTOM gezet. De honderdvierenveertig databytes, vanaf adres 7F27H worden naar het Werkgebied gecopieerd, tussen F380H en F40FH. De functietoetsen worden geïnitialiseerd door de standaardroutine INIFNK. ENDBUF en NLONLY worden nul gemaakt. In BUFMIN wordt een komma gezet en in KBFMIN een dubbele punt. Het adres van de karakterset van de MSX-ROM wordt uit adres 0004H en 0005H gehaald en in CGPNT+1 gezet. PRMPRV wordt geladen met het adres van PRMSTK. In STKTOP, MEMSIZ en VARTAB worden voorlopige

waarden gezet (hun correcte waarde is nog niet bekend), er wordt één In/Out-buffer toegekend (7E6BH) en de Stack Pointer van de Z80 wordt ingevuld (62E5H). In het startadres van RAM wordt een nul-byte gezet, in TXTTAB het adres van het daaropvolgende byte, en er wordt een "NEW"-statement uitgevoerd (6287H).

Nu wordt via de standaardroutines INITIO, INIT32 en CLRSPR de Video Display Processor geïnitialiseerd. De cursor-coördinaten worden op regel 11, kolom 10 ingesteld, en het opstart-bericht "MSX system ..." wordt op het scherm afgedrukt (6678H). Na een pauze van drie seconden worden eventuele uitbreidings-ROMS gezocht (7D75H) en nogmaals een "NEW"-statement uitgevoerd (6287H), voor het geval dat er een BASIC programma in ROM werd uitgevoerd. Tenslotte wordt de identificatietekst "MSX BASIC..." op het scherm gezet (7D29H) en de controle wordt overgedragen naar het "Ok"-punt van de Hoofdlus van de Interpreter (411FH).

adres 7D29H

Deze routine wordt gebruikt tijdens de initialisatieroutine bij het inschakelen, om de tekst van de functietoetsen op het scherm te zetten, het scherm via de standaardroutine INITXT in 40x24 tekstmode te zetten, en de identificatietekst "MSX BASIC ..." op het scherm af te drukken (6678H). Daarna wordt de grootte van het vrije geheugen berekend door de inhoud van VARTAB af te trekken van de inhoud van STKTOP. Het resultaat wordt op het scherm afgedrukt (3412H), gevolgd door de tekst "Bytes free".

adres 7D5DH

Deze routine wordt gebruikt tijdens de initialisatieroutine, om het laagste RAM-adres te zoeken. Vanaf adres EF00H wordt elk byte onderzocht, tot ofwel een byte wordt gevonden waarin niets kan worden geschreven, ofwel adres 8000H is bereikt. Het adres, naar boven afgerond tot het eerstvolgende veelvoud van 256, wordt in registerpaar HL geladen.

adres 7D75H

Deze routine wordt gebruikt tijdens de initialisatie bij het inschakelen van de machine, om te zoeken naar uitbreidings-ROM. Bladzijden 1 en 2 (4000H tot BFFFH) van elk slot worden onderzocht, en het resultaat wordt in SLTATR gezet. Een uitbreidings-ROM bevat op de eerst twee adressen de karakters "AB", waardoor hij van RAM onderscheiden kan worden. Op de eerste zestien adressen staat allerlei informatie over de eigenschappen van de ROM, in deze volgorde :

voorbehouden	byte 10-15
MSB van het adres van een BASIC tekst	byte 9
LSB van het adres van een BASIC tekst	byte 8
MSB van het adres van de apparaat-verwerker	byte 7
LSB van het adres van de apparaat-verwerker	byte 6
MSB van het adres van de statement-verwerker	byte 5
LSB van het adres van de statement-verwerker	byte 4
MSB van het adres van de initialisatieroutine	byte 3
LSB van het adres van de initialisatieroutine	byte 2
42H ("B")	byte 1
41H ("A")	byte 0

Fig. 48 : Informatieblok van een ROM

Elke bladzijde van een bepaald Slot wordt onderzocht. Eerst worden de eerste twee bytes gelezen (7E1AH) en vergeleken met de karakters "AB". Blijkt het om een ROM te gaan, dan wordt het adres van de initialisatie-routine uitgelezen (7E1AH). Via de standaardroutine CALSLT wordt de controle naar dat adres overgedragen. Indien het om een spel in ROM gaat, is het 'best mogelijk dat vanaf dit punt de controle niet meer naar BASIC terugkeert.

Vervolgens wordt het adres gelezen (7E1AH) van de routine die de "CALL"-statements verwerkt die door deze ROM behandeld worden en indien het een geldig adres is, dat wil zeggen verschillend van nul, wordt bit 5 van register B geset. Daarna wordt het adres gelezen (7E1AH) van de routine die extra apparaten behandelt en indien het een geldig adres is, wordt bit 6 van register B geset. Tenslotte wordt het adres gelezen (7E1AH) van het BASIC programma in die ROM en indien dit adres geldig is, wordt bit 7 van register B geset. Tenslotte wordt register B gecopieerd naar de overeenkomstige positie in SLTATR, en de zoektocht gaat verder tot alle slots verwerkt zijn.

Nu wordt in SLTATR gekeken of er ROMs werden gevonden die BASIC tekst bevatten. Zo ja, dan wordt de positie ervan in SLTATR omgerekend in een Slot-identificator (7E2AH) en wordt die ROM, via de standaardroutine ENASLT, permanent ingeschakeld. VARTAB wordt op C000H gezet, omdat de lengte van het BASIC programma niet bekend is. TXTTAB wordt op 8008H gezet en BASROM wordt niet-nul gemaakt, waardoor de "CTRL/STOP"-toets buiten werking wordt gesteld. Het geheugen wordt leeggemaakt (629AH) en de controle gaat over naar de Verwerkingslus (4601H) die het BASIC programma begint uit te voeren.

adres 7E1AH

Deze routine wordt gebruikt om twee opeenvolgende bytes in een uitbreidings-ROM te lezen. In registerpaar HL wordt het eerste adres verwacht en in register C de Slot-identificator. Via de standaardroutine RDSLT worden de bytes ingelezen en in registerpaar DE geladen. Zijn ze beide nul, dan wordt vlag Z geset.

adres 7E2AH

Deze routine berekent uit een positie in SLTATR, die in register B staat, de overeenkomstige Slot-identificator, die in register C wordt geladen, en een ROM-startadres, dat in register H wordt geladen. Vooraf wordt de positie zodanig gewijzigd, dat de volgorde 0->63 wordt in plaats van 64->1. De vereiste informatie staat dan in deze vorm :

7	6	5 4	3 2	1 0
0	0	PSLOT#	SSLOT#	BLZ. NR.

Fig. 49

Bits 0 en 1 worden in de hoogste twee bits van register H geschoven, waar ze een adres vormen. Bits 4 en 5 worden in bits 0 en 1 van register C geschoven, waar ze het Primair Slotnummer aangeven. Bits 2 en 3 worden in bits 2 en 3 van register C geschoven, waar ze het Secundair Slotnummer aangeven. Bit 7 van de overeenkomstige positie in EXPTBL wordt in bit 7 van register C gecopieerd.

adres 7E4BH

Deze routine verwerkt het statement "MAXFILES". De controle wordt naar hier overgedragen, als een "MAX"-token (CDH) wordt gelezen. Eerst wordt gecontroleerd of in de programmatekst een "FILES"-token (B7H) volgt. De operand die het aantal buffers (tussen nul en vijftien) aangeeft, wordt geëvalueerd (521CH) en eventuele bestaande buffers worden gesloten (6C1CH). Het gewenste aantal In/Out-buffers wordt toegekend (7E6BH), het geheugen wordt leeggemaakt (62A7H) en de controle wordt rechtstreeks aan de Verwerkingslus overgedragen (4601H).

adres 7E6BH

Deze routine verzorgt de toekenning van In/Out-buffers. Ze wordt gebruikt door de initialisatieroutine bij het inschakelen van de machine en door de "MAXFILES"- en "CLEAR"-routines, om opslagruimte te voorzien voor het aantal In/Out-buffers dat in register A staat. Voor elke buffer wordt 267 afgetrokken van HIMEM om de nieuwe waarde voor MEMSIZ te verkrijgen. De lengte van het bestaande stringgeheugen (oorspronkelijk tweehonderd bytes) wordt berekend door de vorige inhoud van STKTOP af te trekken van de vorige inhoud van MEMSIZ. Dit resultaat wordt van de nieuwe waarde van MEMSIZ afgetrokken, waardoor de nieuwe waarde voor STKTOP bekend is. Nog eens honderdveertig bytes worden afgetrokken, die voor de stack van de Z80 worden gereserveerd en indien het aldus berekende adres lager is dan de start van het variabelen-geheugen, wordt de foutmelding "Out of memory" gegeven (6275H). In het andere geval wordt het aantal buffers in MAXFIL gezet en worden de nieuwe waarden in MEMSIZ en STKTOP geladen. Het RETURN-adres van de aanroepende routine wordt van de stack gePOPt, de Stack Pointer van de Z80 wordt op de nieuwe positie gezet en het adres wordt terug gePUSHt. In FILTAB wordt het startadres gezet van het blok dat de pointers voor de In/Out-buffers bevat en in elke pointer wordt het adres van het overeenkomstige controleblok gezet. Tenslotte wordt het adres van In/Out-buffer nummer 0 (de "LOAD"- en "SAVE"-buffer van de Interpreter) in NULBUF gezet, en de routine stopt.

adres 7ED8H

Hier staat enkel de tekst "MSX system", gevolgd door een nul-byte.

adres 7EE4H

Hier staat enkel de tekst "version 1.0", plus een "CR"/"LF"-combinatie, gevolgd door een nul-byte.

adres 7EF2H

Hier staat enkel de tekst "MSX BASIC", gevolgd door een nul-byte.

adres 7EFDH

Hier staat enkel de tekst "Copyright 1983 by Microsoft", plus een "CR"/"LF"-combinatie, gevolgd door een nul-byte.

adres 7F1BH

Hier staat enkel de tekst "Bytes free", gevolgd door een nul-byte.

adres 7F27H

Dit blok van honderd vierenveertig bytes wordt gebruikt om het Werkgebied tussen F380H en F40FH te initialiseren.

adres 7FB7H

Dit stukje code van zeven bytes corrigeert een bug in de routine die externe randapparaten verwerkt (55F8H). De routine controleert of de lengte van de naam van het apparaat, die in register A staat, nul is en wijzigt die, zo nodig, in één.

adres 7FBEH

Dit stuk van de ROM wordt niet gebruikt. Het staat vol met nullen.

HOOFDSTUK 6

De geheugenstructuur

De BASIC Interpreter heeft maximaal 32 Kb RAM beschikbaar waarin hij de programmatekst, de BASIC variabelen, de stack van de Z80, de In/Out-buffers en het interne Werkgebied kan opslaan. De tekening hieronder geeft weer hoe dit geheugenblok wordt georganiseerd bij de initialisatie na het inschakelen van de machine.

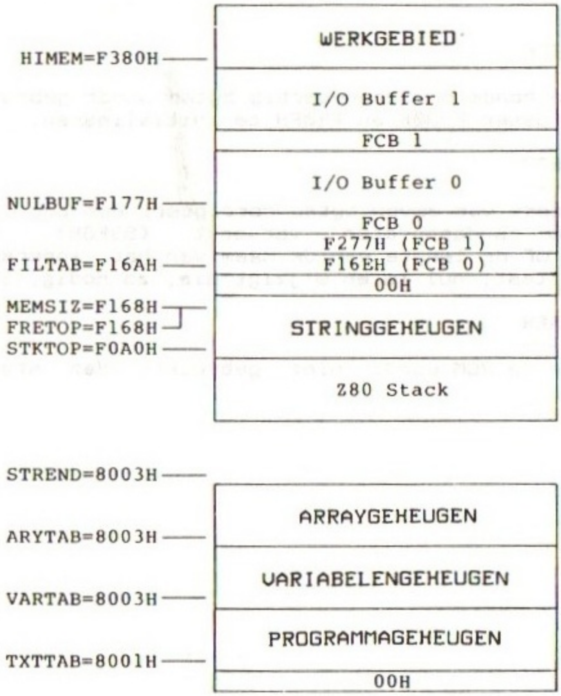


Fig. 50 : Struktuur van het geheugen tussen 8000H en FFFFH

Het Programmageheugen bestaat uit gecodeerde (= van tokens voorziene) programmaregels, in numerieke volgorde volgens de regelnummers, met op het einde het eind-schakeladres 0000H. Na een "NEW" bevat het enkel dit schakeladres. Het nul-byte op adres 8000H is een loze eind-markeerder, die dient om de Verwerkingslus te synchroniseren bij het begin van een programma.

Het Variabelengeheugen en het Arraygeheugen bevatten stringvariabelen, numerieke variabelen en arrays, opgeslagen in de

volgorde waarin ze in de tekst werden gevonden. De snelheid waarmee een programma wordt uitgevoerd, kan een minimale fractie worden opgevoerd door variabelen in een programma toe te wijzen vooraleer arrays worden opgezet, omdat er dan minder geheugeninhoud verplaatst dient te worden.

De stack van de Z80 krijgt een plaats onmiddellijk onder het stringgeheugen. Hij is als volgt gestructureerd :

STKTOP ___	XXH
	00H
Stack Pointer van de Hoofdlus ___	00H
	46H
Stack Pointer van een statement ___	01H

Fig. 51 : de top van de stack van de Z80

Telkens als de stack een andere plaats in het geheugen krijgt, door een "CLEAR"-statement of een "MAXFILES"-statement, worden eerst twee nullen op de stack gePUSHt, als eindmarkeerder wanneer naar een parameterblok wordt gezocht dat bij een "FOR" of een "GOSUB" hoort. In de veronderstelling dat er geen parameterblokken op de stack aanwezig zijn, zal de stackpointer dus (STKTOP-2) bevatten binnen in de Hoofdlus van de Interpreter, en (STKTOP-4) indien de Verwerkingslus de controle overdraagt aan een routine die een statement verwerkt.

Het stringgeheugen bevat de inhoud van de strings die aan variabelen of arrays werden toegekend. Onder die permanente stringstapel kunnen ook een aantal tijdelijke strings aanwezig zijn, tijdens de evaluatie van een uitdrukking. De nul die volgt na het stringgeheugen dient als tijdelijke eindmarkeerder voor de "VAL"-functie.

Het gebied tussen het stringgeheugen en HIMEM wordt gebruikt om de In/Out-buffers in op te slaan. In/Out-buffer nummer nul, de "SAVE"- en "LOAD"-buffer, is altijd in het geheugen aanwezig. Het totale aantal buffers dat ter beschiking van de gebruiker staat, wordt door het statement "MAXFILES" bepaald. Elke In/Out-buffer bestaat uit een file-controleblok, gevolgd door een databuffer van 256 bytes. Het adres van dat blok staat in de tabel onder het file-controleblok nummer 0.

Het file-controleblok bevat de status van de In/Out-buffer, in de volgende vorm :

0	1	2	3	4	5	6	7	8
MOD	00H	00H	00H	DEV	00H	POS	00H	PPS

Fig. 52 : file-controleblok

Het "MOD"-byte bevat de buffer-mode. Het "DEV"-byte bevat de code van het apparaat waaraan de buffer verbonden is. In het "POS"-byte staat de huidige positie in de buffer (0 tot 255), en in het "PPS"-byte de "PRINT"-positie. De rest van het file-controleblok wordt door een standaard MSX-machine niet gebruikt.

HET WERKGEBIED

Het stuk van het Werkgebied tussen F380H en FD99H bevat de variabelen die door het BIOS en de Interpreter worden gebruikt. Op de volgende bladzijden worden ze weergegeven op de manier zoals in Assembler-listings gebruikelijk is.

```
F380H  RDPRIM: OUT  (0A8H),A ;schakel ander Primair Slot in
F382H          LD   E,(HL)   ;lees geheugen
F383H          JR   WRPRM1   ;schakel vorige Primair Slot in
```

Deze routine wordt door de standaardroutine RDSLT gebruikt, om een ander Primair Slot in te schakelen en een byte uit het geheugen te lezen. In register A staat hoe het Primair Slot-register geladen moet worden, in register D wat er voordien in stond. Het gelezen byte wordt in register E geladen.

```
F385H  WRPRIM: OUT  (0A8H),A ;schakel ander Primair Slot in
F387H          LD   (HL),E   ;schrijf naar geheugen
F388H  WRPRM1: LD   A,D      ;haal vorige Slot-instelling terug
F389H          OUT  (0A8H),A ;schakel vorige Primaire Slot in
F38BH          RET
```

Deze routine wordt door de standaardroutine WRSLT gebruikt om een ander Primair Slot in te schakelen en een byte naar het geheugen te schrijven. In register A staat hoe het Primair Slot-register geladen moet worden, in register D wat er voordien in stond. De waarde die in het geheugen moet geschreven worden, staat in register E.

```
F38CH  CLPRIM: OUT   (0A8H),A ;schakel nieuw Primair Slot in
F38EH          EX    AF,AF'   ;naar AF voor CALL
F38FH          CALL  CLPRM1   ;voer uit
F392H          EX    AF,AF'   ;naar AF'
F393H          POP   AF       ;haal vorige instelling terug
F394H          OUT   (0A8H),A ;schakel vorige Primair Slot in
F396H          EX    AF,AF'   ;naar AF
F397H          RET
F398H  CLPRM1: JP    (IX)
```

Deze routine wordt gebruikt door de standaardroutine CALSLT, om een ander Primair Slot in te schakelen en een bepaald adres aan te roepen. In register A staat hoe het Primair Slotregister moet ingesteld worden. De huidige instelling staat op de stack van de Z80. Het adres dat aangeroepen moet worden, staat in registerpaar IX.

```
F39AH  USRTAB: DEFW  475AH    ; USR 0
F39CH          DEFW  475AH    ; USR 1
F39EH          DEFW  475AH    ; USR 2
F3A0H          DEFW  475AH    ; USR 3
F3A2H          DEFW  475AH    ; USR 4
F3A4H          DEFW  475AH    ; USR 5
F3A6H          DEFW  475AH    ; USR 6
F3A8H          DEFW  475AH    ; USR 7
F3AAH          DEFW  475AH    ; USR 8
F3ACH          DEFW  475AH    ; USR 9
```

Deze tien variabelen bevatten de adressen van de "USR"-functies. Bij het inschakelen van de machine krijgen ze de waarde 475AH, dat is het adres waar de Interpreter de foutmelding "Illegal function call" produceert. Nadien worden ze enkel nog door het "DEFUSR"-statement gewijzigd.

```
F3AEH  LINL40: DEFB 37
```

Deze variabele bevat de schermbreedte in de 40x24 tekstmode. Hij krijgt zijn waarde tijdens de initialisatie bij het inschakelen, en wordt nadien enkel nog gewijzigd door het "WIDTH"-statement.

```
F3AFH  LINL32: DEFB 29
```

Deze variabele bevat de schermbreedte in 32x24 tekstmode. Hij krijgt zijn waarde tijdens de initialisatie bij het inschakelen, en wordt nadien enkel nog gewijzigd door het "WIDTH"-statement.

```
F3B0H  LINLEN: DEFB 37
```

Deze variabele bevat de schermbreedte in de huidige tekstmode. Telkens als de Video Display Processor via de standaardroutines INITXT of INIT32 voor een van beide tekstmodes wordt geïnitialiseerd, krijgt deze variabele de waarde die respectievelijk in LINL40 of LINL32 staat.

```
F3B1H  CRTCNT: DEFB 24
```

Deze variabele bevat het aantal regels op het scherm. Hij krijgt zijn waarde bij de initialisatie na het inschakelen, en wordt nadien niet meer gewijzigd.

```
F3B2H  CLMLST: DEFB 14
```

Deze variabele geeft aan hoeveel kolommen minimaal nog op een regel vrij moeten zijn, om een data-item te "PRINT"-en. Is er minder ruimte, dan wordt eerst een "CR"/"LF"-combinatie uitgevoerd. Deze variabele krijgt zijn waarde tijdens de initialisatie na het inschakelen en wordt nadien enkel nog gewijzigd door de "WIDTH"- en "SCREEN"-statements.

```
F3B3H  TXTNAM: DEFW 0000H ;startadres van de Namentabel
F3B5H  TXTCOL: DEFW 0000H ;startadres van Kleurentabel
F3B7H  TXTCGP: DEFW 0800H ;startadres van Karakterpatronen
F3B9H  TXTATR: DEFW 0000H ;startadres van Sprite-attributen
F3BBH  TXTPAT: DEFW 0000H ;startadres van Sprite-patronen
```

Deze vijf variabelen bevatten de basisadressen van de tabellen die de Video Display Processor gebruikt in de 40x24 tekstmode. Ze krijgen hun waarde tijdens de initialisatie na het inschakelen en worden nadien enkel nog gewijzigd door het "BASE"-statement.

```
F3BDH  T32NAM: DEFW 1800H ;startadres van de Namentabel
F3BFH  T32COL: DEFW 2000H ;startadres van Kleurentabel
F3C1H  T32CGP: DEFW 0000H ;startadres van Karakterpatronen
F3C3H  T32ATR: DEFW 1B00H ;startadres van Sprite-attributen
F3C5H  T32PAT: DEFW 3800H ;startadres van Sprite-patronen
```

Deze vijf variabelen bevatten de basisadressen van de tabellen

die de Video Display Processor gebruikt in de 32x24 tekstmode. Ze krijgen hun waarde tijdens de initialisatie na het inschakelen en worden nadien enkel nog gewijzigd door het "BASE"-statement.

```
F3C7H  GRPNAM: DEFW 1800H ;startadres van de Namentabel
F3C9H  GRPCOL: DEFW 2000H ;startadres van Kleurentabel
F3CBH  GRPCGP: DEFW 0000H ;startadres van Karakterpatronen
F3CDH  GRPATR: DEFW 1B00H ;startadres van Sprite-attributen
F3CFH  GRPPAT: DEFW 3800H ;startadres van Sprite-patronen
```

Deze vijf variabelen bevatten de basisadressen van de tabellen die de Video Display Processor gebruikt in de grafische mode. Ze krijgen hun waarde tijdens de initialisatie na het inschakelen en worden nadien enkel nog gewijzigd door het "BASE"-statement.

```
F3D1H  MLTNAM: DEFW 0800H ;startadres van de Namentabel
F3D3H  MLTCOL: DEFW 0000H ;startadres van Kleurentabel
F3D5H  MLTCGP: DEFW 0000H ;startadres van Karakterpatronen
F3D7H  MLTATR: DEFW 1B00H ;startadres van Sprite-attributen
F3D9H  MLTPAT: DEFW 3800H ;startadres van Sprite-patronen
```

Deze vijf variabelen bevatten de basisadressen van de tabellen die de Video Display Processor gebruikt in de veelkleurenmode. Ze krijgen hun waarde tijdens de initialisatie na het inschakelen en worden nadien enkel nog gewijzigd door het "BASE"-statement.

```
F3DBH  CLIKSW: DEFB 01H
```

De interrupt-verwerker bepaalt aan de hand van de inhoud van deze variabele of hij bij het indrukken van een toets wel (inhoud verschilt van nul) of niet (inhoud is nul) een klik moet produceren. Bij de initialisatie na het inschakelen krijgt hij de waarde 1. Nadien wordt hij enkel nog gewijzigd door het "SCREEN"-statement.

```
F3DCH  CSRY:   DEFB 01H
```

Deze variabele bevat een waarde tussen 1 en CRTCNT. Die geeft het regel-coördinaat aan van de cursor in tekstmode.

```
F3DDH  CSRX:   DEFB 01H
```

Deze variabele bevat een waarde tussen 1 en LINLEN. Die geeft het kolom-coördinaat aan van de cursor in tekstmode. Merk op dat de coördinaten van de cursor voor de "HOME"-positie door het BIOS altijd op 1,1 gehouden worden, wat de schermbreedte ook is.

```
F3DEH  CNSDFG: DEFB FFH
```

De inhoud van deze variabele bepaalt of de tekst van de functietoetsen wel (inhoud verschilt van nul) of niet (inhoud is nul) op het scherm worden afgedrukt.

```
F3DFH  RG0SAV: DEFB 00H
F3E0H  RG1SAV: DEFB F0H
F3E1H  RG2SAV: DEFB 00H
F3E2H  RG3SAV: DEFB 00H
F3E3H  RG4SAV: DEFB 01H
F3E4H  RG5SAV: DEFB 00H
F3E5H  RG6SAV: DEFB 00H
F3E6H  RG7SAV: DEFB F4H
```

Deze acht variabelen zijn een copie van de acht moderegisters van de Video Display Processor, waarin enkel geschreven kan worden. De getoonde waarden gelden voor de 40x24 tekstmode.

```
F3E7H  STATFL: DEFB CAH
```

In deze variabele wordt bij elke interrupt de inhoud van het Statusregister van de Video Display Processor gezet.

```
F3E8H  TRGFLG: DEFB F1H
```

In deze variabele wordt door de interrupt-routines continu de toestand van de vier vuurknop-inputs en de spatietoets gezet.

```
F3E9H  FORCLR: DEFB 0FH      ; wit
```

Deze variabele bevat de huidige voorgrondkleur. Bij de initialisatie wordt er de waarde 0FH in gezet, en die wordt nadien enkel door het "COLOR"-statement gewijzigd. De voorgrondkleur wordt door de standaardroutine CLRSPR gebruikt, om de kleur van de sprites in te stellen. Ze wordt eveneens door de standaardroutine CHGCLR gebruikt om in de tekstmodes de kleur van de "1"-pixels te bepalen. Deze kleur is ook de inktkleur voor de grafische modes, omdat ze door de standaardroutine GRPPRT in ATRBYT gecopieerd wordt, en omdat ze door alle routines van de Interpreter als standaardwaarde gebruikt wordt als een of andere facultatieve kleur-operand niet wordt vermeld.

```
F3EAH  BAKCLR: DEFB 04H      ; donkerblauw
```

Deze variabele bevat de huidige achtergrondkleur. Bij de initialisatie wordt er de waarde 04H in gezet, en die wordt nadien enkel nog door het "COLOR"-statement gewijzigd. De achtergrondkleur wordt door de standaardroutine CLS gebruikt om, in de grafische modes, het scherm te wissen. Ze wordt eveneens gebruikt door de standaardroutine CHGCLR om in de tekstmodes de kleur van de "0"-pixels te bepalen.

```
F3EBH  BDRCLR: DEFB 04H      ; donkerblauw
```

Deze variabele bevat de huidige kleur van de rand van het scherm. Bij de initialisatie wordt er de waarde 04H in gezet en die wordt nadien enkel door het statement "COLOR" veranderd. De kleur van de border wordt gebruikt door de standaardroutine CHGCLR in de 32x24 tekstmode, de grafische mode en de veelkleurenmode, om de kleur van de rand van het scherm in te stellen.

```
F3ECH  MAXUPD: DEFB C3H
F3EDH          DEFW 0000H
```

De laatste twee bytes worden door de "LINE"-routine ingevuld met een adres, waardoor de groep van drie bytes voor de Z80 een JP-instructie vormt naar één van de standaardroutines RIGHTC, LEFTC, UPC of DOWNC.

```
F3EFH  MINUPD: DEFB C3H
F3F0H          DEFW 0000H
```

De laatste twee bytes worden door de "LINE"-routine ingevuld met een adres, waardoor de groep van drie bytes voor de Z80 een JP-instructie vormt naar één van de standaardroutines RIGHTC, LEFTC, UPC of DOWNC.

```
F3F2H  ATRBYT: DEFB 0FH
```

Deze variabele bevat de inktkleur voor de grafische modes, die wordt gebruikt door de standaardroutines SETC en NSETCX.

```
F3F3H  QUEUES: DEFB F959H
```

Deze variabele bevat het adres van de controleblokken voor de drie muziekrijen. De waarde die er bij de initialisatie in wordt gezet, wijzigt nadien niet meer.

```
F3F5H  FRCNEW: DEFB FFH
```

Deze variabele wordt als een vlag gebruikt, om het onderscheid te maken tussen "CLOAD" (00H) en "CLOAD?" (FFH).

```
F3F6H  SCNCNT: DEFB 01H
```

Deze variabele wordt als een teller gebruikt door de interrupt-routines, om de frekwentie te bepalen waarmee het toetsenbord moet worden afgetast.

```
F3F7H  REPCNT: DEFB 01H
```

Deze variabele wordt door de interrupt-routines als een teller gebruikt, om te bepalen met welke snelheid de toetsen repeteren.

```
F3F8H  PUTPNT: DEFW FBF0H
```

In deze variabele staat het adres van de eerstvolgende positie in KEYBUF waar een karakter geplaatst kan worden.

```
F3FAH  GETPNT: DEFW FBF0H
```

In deze variabele staat het adres van de eerstvolgende positie in KEYBUF waar een karakter opgehaald kan worden.

```
F3FCH  CS1200: DEFB 53H ; LAAG cyclus 1ste helft
F3FDH          DEFB 5CH ; LAAG cyclus 2de helft
F3FEH          DEFB 26H ; HOOG cyclus 1ste helft
F3FFH          DEFB 2DH ; HOOG cyclus 2de helft
F400H          DEFB 0FH ; aantal cycli in aanloopblok
```

Deze vijf variabelen bevatten de parameters om met 1200 baud naar cassette te schrijven. Ze krijgen hun waarde tijdens de initialisatie na het inschakelen van de machine en worden nadien niet meer gewijzigd.

```
F401H  CS2400: DEFB 25H ; LAAG cyclus 1ste helft
F402H          DEFB 2DH ; LAAG cyclus 2de helft
F403H          DEFB 0EH ; HOOG cyclus 1ste helft
F404H          DEFB 16H ; HOOG cyclus 2de helft
F405H          DEFB 1FH ; aantal cycli in aanloopblok
```

Deze vijf variabelen bevatten de parameters om met 2400 baud naar cassette te schrijven. Ze krijgen hun waarde tijdens de initialisatie na het inschakelen van de machine en worden nadien niet meer gewijzigd.

```
F406H     LOW: DEFB 53C ; LAAG cyclus 1ste helft
F407H          DEFB 5CH ; LAAG cyclus 2de helft
F408H    HIGH: DEFB 26H ; HOOG cyclus 1ste helft
F409H          DEFB 2DH ; HOOG cyclus 2de helft
F40AH  HEADER: DEFB 0FH ; aantal cycli in aanloopblok
```

Deze vijf variabelen bevatten de parameters die op dit ogenblik gebruikt worden om naar cassette te schrijven. Ze krijgen de waarden voor 1200 baud, bij de initialisatie na het inschakelen van de machine, en worden nadien enkel nog gewijzigd door het "CSAVE"- en "SCREEN"-statement.

```
F40BH  ASPCT1: DEFW 0100H
```

Deze variabele bevat de reciproke van de standaardwaarde van de verhouding tussen de stralen van een "CIRCLE", vermenigvuldigd met 256. Hij krijgt die waarde bij het inschakelen van de machine en wordt nadien niet meer gewijzigd.

```
F40DH  ASPCT2: DEFW 0100H
```

Deze variabele bevat de standaardwaarde van de verhouding tussen de stralen van een "CIRCLE", vermenigvuldigd met 256. Hij krijgt die waarde bij het inschakelen van de machine en wordt nadien niet meer gewijzigd. Die verhouding staat in twee gedaanten ter beschikking, zodanig dat de "CIRCLE"-routine één van beide rechtstreeks kan selecteren zonder verdere controle of mogelijke bewerking om een waarde om te keren, wat het geval is als in de programmatekst een operand volgt op het statement.

```
F40FH  ENDPRG: DEFB ":"
F410H          DEFB 00H
F411H          DEFB 00H
F412H          DEFB 00H
F413H          DEFB 00H
```

Deze vijf bytes vormen een loze programmaregel. Ze krijgen hun waarde bij het inschakelen van de machine en worden nadien niet meer gewijzigd. Die regel wordt gebruikt, indien in de Hoofdlus van de Interpreter een fout ontdekt wordt voordat er in KBUF gecodeerde programmatekst aanwezig is. Indien op dat moment een "ON ERROR GOTO"-statement in werking is, dan kan het "RESUME"-statement op deze regel eindigen.

F414H ERRFLG: DEFB 00H

Deze variabele wordt door de foutverwerkingsroutine van de Interpreter gebruikt om het nummer van de ontdekte fout in op te slaan.

F415H LPTPOS: DEFB 00H

Deze variabele wordt door de "LPRINT"-routine gebruikt om de positie van de printerkop bij te houden.

F416H PRTFLG: DEFB 00H

De inhoud van deze variabele bepaalt of de standaardroutine OUTDO output naar het scherm (00H) of de printer (01H) stuurt.

F417H NTMSXP: DEFB 00H

De inhoud van deze variabele wordt door de standaardroutine OUTDO gebruikt, wanneer grafische karakters, voorafgegaan door een grafische header, naar de printer gestuurd worden. Is de variabele 0, dan worden ze als grafische karakters gestuurd. Verschilt hij van nul, dan worden die karakters vervangen door spaties. De variabele krijgt zijn waarde bij het aanschakelen van de machine en wordt nadien enkel nog gewijzigd door het "SCREEN"-statement.

F418H RAWPRT: DEFB 00H

De inhoud van deze variabele wordt door de standaardroutine OUTDO gebruikt, wanneer grafische karakters, voorafgegaan door een grafische header of controlekarakters naar de printer gestuurd worden. Is de variabele 0, dan worden ze gewijzigd. Verschilt hij van nul, dan worden ze gestuurd zoals ze zijn. De variabele krijgt zijn waarde bij het inschakelen van de machine en wordt nadien niet meer gewijzigd.

F419H VLZADR: DEFW 0000H
F41BH VLZDAT: DEFB 00H

Deze variabelen bevatten het adres en de waarde van een karakter dat tijdelijk door de "VAL"-functie werd weggehaald.

F41CH CURLIN: DEFW FFFFH

Deze variabele bevat het regelnummer dat de Interpreter momenteel verwerkt. De waarde FFFFH geeft aan dat hij in directe mode werkt.

F41EH KBFMIN: DEFB ":"

Dit byte dient als loos voorvoegsel van de gecodeerde tekst in KBUF. Het heeft een gelijkaardige functie als ENDPRG : het wordt gebruikt indien een fout ontdekt wordt bij de verwerking van een direct statement.

F41FH KBUF: DEFS 318

Deze buffer bevat de gecodeerde vorm van de ingetypte regel die door de Hoofdlus van de Interpreter werd ingelezen. Indien een

direct statement wordt uitgevoerd, vormt de inhoud van deze buffer de programmatekst.

F55DH BUFMIN: DEFB ","

Dit byte dient als loos voorvoegsel van de tekst in BUF. Het wordt gebruikt om de "INPUT"-routine te synchroniseren, wanneer die de tekst van de input begint te analyseren.

F55EH BUF: DEFS 259

Deze buffer bevat de tekst die door de standaardroutine INLIN van het toetsenbord werd ingelezen.

F661H TTYPOS: DEFB 00H

Deze variabele wordt door de "PRINT"-routine gebruikt om de positie op het scherm bij te houden (Telex!).

F662H DIMFLG: DEFB 00H

Deze variabele bevat normaal nul. Zijn inhoud wordt gewijzigd door de "DIM"-routine, om de werking van de variabelen-zoekroutine te sturen.

F663H VALTYP: DEFB 02H

Deze variabele bevat de code van het type variabele dat momenteel in DAC staat : 2 voor een geheel getal, 3 voor een string, 4 voor enkele precisie en 8 voor dubbele precisie.

F664H DORES: DEFB 00H

Deze variabele bevat normaal nul. Zijn inhoud wordt gewijzigd om aan te geven dat keywords die niet tussen aanhalingstekens staan, na een "DATA"-token, niet in tokens omgezet moeten worden.

F665H DONUM: DEFB 00H

Deze variabele bevat normaal nul. Zijn inhoud wordt gewijzigd indien er na een van de keywords "GOTO", "GOSUB", "THEN" enzovoort, een numerieke constante volgt die omgezet dient te worden in de kenmerkende vorm van een regelnummer-operand.

F666H CONTXT: DEFW 0000H

Deze variabele wordt door de standaardroutine CHRGTR gebruikt, om het adres in op te slaan van het karakter, dat in de programmatekst volgt op een numerieke constante.

F668H CONSAV: DEFB 00H

Deze variabele wordt door de standaardroutine CHRGTR gebruikt om het token in op te slaan van een numerieke constante, die in de programmatekst werd gevonden.

F669H CONTYP: DEFB 00H

Deze variabele wordt door de standaardroutine CHRGTR gebruikt om het type in op te slaan van een numerieke constante, die in de programmatekst werd gevonden.

F66AH CONLO: DEFS 8

Deze variabele wordt door de standaardroutine CHRGTR gebruikt om de waarde in op te slaan van een numerieke constante, die in de programmatekst werd gevonden.

F672H MEMSIZ: DEFW F168H

Deze variabele bevat het adres van de top van het stringgeheugen. Hij krijgt zijn waarde bij het inschakelen van de machine en wordt nadien enkel nog gewijzigd door een "CLEAR"- of een "MAXFILES"-statement.

F674H STKTOP: DEFW F0A0H

Deze variabele bevat het adres van de top van de stack van de Z80. Hij krijgt zijn waarde (MEMSIZ - 200) bij het inschakelen van de machine en wordt nadien enkel nog gewijzigd door de statements "CLEAR" en "MAXFILES".

F676H TXTTAB: DEFW 8001H

Deze variabele bevat het adres van het eerste byte van de programmageheugen. Hij krijgt zijn waarde bij het inschakelen van de machine en wordt nadien niet meer gewijzigd.

F678H TEMPPT: DEFW F67AH

Deze variabele bevat het adres van de eerstvolgende vrije positie in TEMPST.

F67AH TEMPST: DEFS 30

Deze buffer wordt gebruikt om signalementen van strings in op te slaan. Hij werkt als een stack : routines die strings produceren, PUSHen een signalement en routines die strings gebruiken, POPpen het.

F698H DSCTMP: DEFS 3

Deze buffer wordt door de string-functies gebruikt om een signalement tijdelijk in op te slaan, tijdens de opmaak van het signalement.

F69BH FRETOP: DEFW F168H

Deze variabele bevat het adres van de eerstvolgende vrije positie in het stringgeheugen. Is dit leeg, dan is FRETOP gelijk aan MEMSIZ.

F69DH TEMP3: DEFW 0000H

Deze variabele wordt door diverse onderdelen van de Interpreter gebruikt als tijdelijke opslagruimte.

F69FH TEMP8: DEFW 0000H

Deze variabele wordt door diverse onderdelen van de Interpreter gebruikt als tijdelijke opslagruimte.

F6A1H ENDFOR: DEFW 0000H

Deze variabele wordt door de "FOR"-routine gebruikt om, tijdens het aanmaken van een parameterblok, het eindadres van het statement in op te slaan.

F6A3H DATLIN: DEFW 0000H

Deze variabele bevat het regelnummer waarop het "DATA"-item staat dat momenteel wordt verwerkt.

F6A5H SUBFLG: DEFB 00H

Deze variabele bevat normaal nul. Zijn inhoud wordt gewijzigd door de routines die "ERASE", "FOR", "FN" en "DEF FN" verwerken, om te regelen hoe indexen van variabelen verwerkt worden door de variabelen-zoekroutine.

F6A6H FLGINP: DEFB 00H

Deze variabele dient als vlag voor de routine die de statements "READ" en "INPUT" verwerkt, om uit te maken welke van beide het is : "INPUT" (variabele is nul) of "READ" (variabele verschilt van nul).

F6A7H TEMP: DEFW 0000H

Deze variabele wordt door diverse onderdelen van de Interpreter gebruikt als tijdelijke opslagruimte.

F6A9H PTRFLG: DEFB 00H

Deze variabele bevat normaal nul. Zijn inhoud wordt gewijzigd op het moment dat er een regelnummer in de programmatekst, in een adres-pointer wordt omgezet.

F6AAH AUTFLG: DEFB 00H

Deze variabele bevat normaal nul. Zijn inhoud wordt gewijzigd als het "AUTO"-statement in werking wordt gesteld.

F6ABH AUTLIN: DEFW 0000H

Deze variabele bevat het huidige regelnummer, tijdens het verwerken van het "AUTO"-statement.

F6ADH AUTINC: DEFW 0000H

Deze variabele bevat de regelafstand ("STEP"-waarde) die door het "AUTO" - statement in acht genomen moet worden.

F6AFH SAVTXT: DEFW 0000H

Bij het begin van elk statement zet de Verwerkingslus in deze variabele het adres uit de programmatekst waar de Interpreter

gekomen is. De variabele wordt gebruikt tijdens het verwerken van fouten, om ERRTXT klaar te maken voor de "RESUME"-routine en OLDTXT voor de "CONT"-routine.

F6B1H SAVSTK: DEFW F09EH

Bij het begin van elk statement zet de Verwerkingslus in deze variabele de inhoud van de stackpointer van de Z80, waar de foutverwerker hem kan uitlezen.

F6B3H ERRLIN: DEFW 0000H

Deze variabele wordt door de foutverwerkingsroutine gebruikt om het regelnummer in te zetten van de programmaregel waarin een fout optreedt.

F6B5H DOT: DEFW 0000H

In deze variabele wordt door de Hoofdlus en door de foutverwerkingsroutine het nummer van de laatst behandelde regel gezet, waar het opgehaald wordt als in een statement het "."-teken als parameter wordt gebruikt.

F6B7H ERRTXT: DEFW 0000H

Op deze plaats zet de foutverwerkingsroutine de inhoud van SAVTXT, waar hij door de "RESUME"-routine weer opgehaald kan worden.

F6B9H ONELIN: DEFW 0000H

Op deze plaats zet de "ON ERROR GOTO"-routine het adres van de programmaregel die uitgevoerd moet worden, indien een fout optreedt.

F6BBH ONEFLG: DEFB 00H

Deze variabele bevat normaal nul. De foutverwerkingsroutine wijzigt zijn inhoud, op het ogenblik dat de controle overgedragen wordt aan een "ON ERROR GOTO"-statement. Hiermee wordt voorkomen dat een lus-mechanisme opgebouwd wordt indien in de statements die de foutverwerking afhandelen, ook een fout zou optreden.

F6BCH TEMP2: DEFW 0000H

Deze variabele wordt door diverse onderdelen van de Interpreter gebruikt als tijdelijke opslagruimte.

F6BEH OLDLIN: DEFW 0000H

Deze variabele bevat het nummer van de regel waarop het programma stopte. Hij krijgt die inhoud door de "END"-routine of door de "STOP"-routine. Een "CONT"-statement zal hiervan gebruik maken om het programma voort te zetten.

F6C0H OLDTXT: DEFW 0000H

Deze variabele bevat het adres van het statement waarmee het programma beëindigd werd.

F6C2H VARTAB: DEFW 8003H

Deze variabele bevat het adres van het eerste byte in het variabelengeheugen.

F6C4H ARYTAB: DEFW 8003H

Deze variabele bevat het adres van het eerste byte in het arraygeheugen.

F6C6H STREND: DEFW 8003H

Deze variabele bevat het adres van het byte, dat volgt op het einde van het arraygeheugen.

F6C8H DATPTR: DEFW 8000H

Deze variabele bevat het adres in de programmatekst van het "DATA"-item dat momenteel wordt gebruikt.

```
F6CAH  DEFTBL: DEFB 08H ; A
F6CBH          DEFB 08H ; B
F6CCH          DEFB 08H ; C
F6CDH          DEFB 08H ; D
F6CEH          DEFB 08H ; E
F6CFH          DEFB 08H ; F
F6D0H          DEFB 08H ; G
F6D1H          DEFB 08H ; H
F6D2H          DEFB 08H ; I
F6D3H          DEFB 08H ; J
F6D4H          DEFB 08H ; K
F6D5H          DEFB 08H ; L
F6D6H          DEFB 08H ; M
F6D7H          DEFB 08H ; N
F6D8H          DEFB 08H ; O
F6D9H          DEFB 08H ; P
F6DAH          DEFB 08H ; Q
F6DBH          DEFB 08H ; R
F6DCH          DEFB 08H ; S
F6DDH          DEFB 08H ; T
F6DEH          DEFB 08H ; U
F6DFH          DEFB 08H ; V
F6E0H          DEFB 08H ; W
F6E1H          DEFB 08H ; X
F6E2H          DEFB 08H ; Y
F6E3H          DEFB 08H ; Z
```

Deze zesentwintig variabelen bevatten het standaard-type voor elke groep BASIC variabelen. Bij het inschakelen, en eveneens door een "NEW" of een "CLEAR"-statement, wordt hun waarde ingesteld als "dubbele precisie". Nadien wordt die enkel gewijzigd door de groep "DEF"-statements.

F6E4H PRMSTK: DEFW 0000H

Deze variabele bevat het startadres van het vorige parameterblok van een "FN"-statement op de stack van de Z80. Hij wordt gebruikt tijdens de "garbage collection" om op de stack van blok tot blok te springen.

F6E6H PRMLEN: DEFW 0000H

Deze variabele bevat de lengte van het "FN"-parameterblok dat momenteel in PARM1 staat.

F6E8H PARM1: DEFS 100

Deze buffer bevat de locale variabelen die bij de "FN"-functie horen die op een bepaald moment geëvalueerd worden.

F74CH PRMPRV: DEFW F6E4H

Deze variabele bevat het adres van het vorige parameterblok van een "FN"-statement. Het is in feite een constante, die ervoor moet voor zorgen dat de "garbage collection" altijd begint met het huidige parameterblok, vooraleer de blokken op de stack doorzocht worden.

F74EH PRMLN2: DEFW 0000H

Deze variabele bevat de lengte van het "FN"-parameterblok dat in PARM2 geconstrueerd wordt.

F750H PARM2: DEFS 100

In deze buffer worden de locale variabelen geconstrueerd, die horen bij de "FN"-functie die op dat moment wordt geëvalueerd.

F7B4H PRMFLG: DEFB 00H

Deze variabele dient als vlag voor de variabelen-zoekroutine, om aan te geven of de variabelen die onderzocht worden, locale dan wel globale variabelen zijn.

F7B5H ARYTA2: DEFW 0000H

Op deze plaatsen wordt tijdens het zoeken naar een variabele, het laatste adres gezet van het geheugenblok waarin gezocht wordt (variabelengeheugen, arraygeheugen, ...).

F7B7H NOFUNS: DEFB 00H

Normaal bevat deze variabele nul. Zijn inhoud wordt gewijzigd door de "FN"-functieroutine, om aan de variabelen-zoekroutine te signaleren dat er locale variabelen bestaan.

F7B8H TEMP9: DEFW 0000H

Deze variabele wordt door diverse onderdelen van de Interpreter gebruikt als tijdelijke opslagruimte.

F7BAH FUNACT: DEFW 0000H

In deze variabele wordt het aantal "FN"-functies opgeslagen die op dat ogenblik in werking zijn.

F7BCH SWPTMP: DEFS 8

Deze buffer dient om de eerste operand in een "SWAP"-statement in op te slaan.

F7C4H TRCFLG: DEFB 00H

Normaal bevat deze variabele nul. De "TRACE"-routine wijzigt zijn inhoud, om de "trace"-functie in werking te zetten.

F7C5H FBUFFR: DEFS 43

In deze buffer wordt de tekst gezet, die tijdens de omzetting van numerieke output geproduceerd wordt.

F7F0H DECTMP: DEFW 0000H

Deze variabele wordt door de routine die getallen met dubbele precisie deelt, gebruikt als tijdelijke opslagruimte.

F7F2H DECTM2: DEFW 0000H

Deze variabele wordt door de routine die twee getallen met dubbele precisie deelt, gebruikt als tijdelijke opslagruimte.

F7F4H DECCNT: DEFB 00H

Deze variabele wordt door de routine die twee getallen met dubbele precisie deelt, gebruikt om het aantal van nul verschillende bytes in de mantisse van de tweede operand te bevatten.

F7F6H DAC: DEFS 16

Deze buffer dient als primaire accumulator voor de Interpreter, tijdens de evaluatie van uitdrukkingen.

F806H HOLD8: DEFS 65

Deze buffer wordt door de routine die twee getallen met dubbele precisie vermenigvuldigt, gebruikt om de veelvouden van de eerste operand in op te slaan.

F847H ARG: DEFS 16

Deze buffer dient als secundaire accumulator voor de Interpreter, tijdens de evaluatie van uitdrukkingen.

F857H RNDX: DEFS 8

In deze buffer staat het huidige toevalsgetal (dubbele precisie).

F85FH MAXFIL: DEFB 01H

Deze variabele bevat het aantal In/Out-buffers dat momenteel gebuikt kan worden. Bij het inschakelen wordt zijn waarde 1. Nadien wordt die enkel nog gewijzigd door het statement "MAXFILES".

F860H FILTAB: DEFW F16AH

Deze variabele bevat het adres van de tabel met pointers voor de file-controleblokken van de In/Out-buffers.

F862H NULBUF: DEFW F177H

Deze variabele bevat het adres van het eerste byte van de databuffer die bij In/Out-buffer nummer 0 hoort.

F864H PTRFIL: DEFW 0000H

Deze variabele bevat het adres van het file-controleblok van de In/Out-buffer die op dit ogenblik in werking is.

F866H FILNAM: DEFS 11

In deze buffer staat de filenaam die de gebruiker opgaf. De buffer kan elf karakters bevatten, waardoor ook namen van files voor disk-drive, zoals "FILENAME.BAS", verwerkt kunnen worden.

F871H FILNM2: DEFS 11

In deze buffer wordt de file-naam gezet die van een In/Out-apparaat werd gelezen, om hem te vergelijken met de inhoud van FILNAM.

F87CH NLONLY: DEFB 00H

Normaal is deze variabele nul. Zijn inhoud wordt gewijzigd wanneer een programma ingeladen wordt via het "LOAD"-statement. Bit 0 vermijdt dat In/Out-buffer nummer 0 gesloten wordt tijdens het inladen. Bit 7 vermijdt dat de In/Out-buffers, die door de gebruiker werden gedefinieerd, worden gesloten, als het programma automatisch moet opstarten na het laden.

F87DH SAVEND: DEFW 0000H

In deze variabele zet de "BSAVE"-routine het eindadres van het geheugenblok dat ge"SAVE"d moet worden.

F87FH FNKSTR: DEFS 160

Deze buffer bevat de tien strings van zestien karakters, die bij de functietoetsen horen. Ze worden ingevuld tijdens de initialisatie na het inschakelen van de machine. Hun inhoud wordt nadien enkel nog gewijzigd door het "KEY"-statement.

```
F91FH   CGPNT: DEFB 00H   ; Slot-identificator
F920H          DEFW 1BBFH ; adres
```

Deze variabelen bevatten de plaats in het geheugen waar de karakerset staat die door de standaardroutines INITXT en INIT32 naar de Video Display Processor worden gecopieerd. Bij het inschakelen worden hier de waarden ingevuld voor de karakterset die in de MSX ROM staat. Die worden nadien niet meer gewijzigd.

F922H NAMBAS: DEFW 0000H

In deze variabele staat het startadres van de Namentabel in de Video Display Processor, voor de huidige tekstmode. Telkens wanneer de Video Display Processor door de standaardroutines INITXT of INIT32 voor een bepaalde tekstmode wordt geïnitialiseerd, wordt hier de inhoud van respectievelijk TXTNAM of T32NAM ingevuld.

F924H CGPBAS: DEFW 0800H

In deze variabele staat het startadres van de Karaktertabel in de Video Display Processor, voor de huidige tekstmode. Telkens wanneer de Video Display Processor door de standaardroutines INITXT of INIT32 voor een bepaalde tekstmode wordt geïnitialiseerd, wordt hier de inhoud van respectievelijk TXTCGP of T32CGP ingevuld.

F926H PATBAS: DEFW 3800H

In deze variabele staat het startadres van de momenteel gebruikte Tabel met Sprite-patronen in de Video Display Processor. Telkens wanneer de Video Display Processor door de standaardroutines INIT32, INIGRP of INIMLT wordt geïnitialiseerd, wordt hier de inhoud van respectievelijk T32PAT, GRPPAT of MLTPAT ingevuld.

F928H ATRBAS: DEFW 1B00H

In deze variabele staat het startadres van de momenteel gebruikte Tabel met Sprite-attributen in de Video Display Processor. Telkens wanneer de Video Display Processor door de standaardroutines INIT32, INIGRP of INIMLT wordt geïnitialiseerd, wordt hier de inhoud van respectievelijk T32ATR, GRPATR of MLTATR ingevuld.

```
F92AH   CLOC: DEFW 0000H ; positie van de pixel
F92CH  CMASK: DEFB 80H   ; pixel-masker
```

Deze variabelen bevatten de scherm-positie van de laatst verwerkte pixel. Die wordt gebruikt door de standaardroutines RIGHTC, LEFTC, UPC, TUPC, DOWNC, TDOWNC, FETCHC, STOREC, READC, SETC, NSETCX, SCANR en SCANR. In CLOC staat het adres van het byte waarin de pixel zich bevindt, en CMASK geeft de positie van de pixel in dat byte aan.

F92DH MINDEL: DEFW 0000H

Deze variabele wordt door de "LINE"-routine gebruikt om het minimaal verschil in op te slaan tussen de eindpunten van een lijn.

F92FH MAXDEL: DEFW 0000H

Deze variabele wordt door de "LINE"-routine gebruikt om het maximaal verschil tussen de eindpunten van een lijn in op te slaan.

F931H ASPECT: DEFW 0000H

In deze variabele slaat de "CIRCLE"-routine de verhouding tussen de stralen van een cirkel op. Dit gebeurt in de vorm van een binaire fractie van één byte, zodat een verhouding 0,75 zou worden opgeslagen als 00C0H. Het MSB is enkel nodig als de verhouding precies 1,00 is : dat wordt dan 0100H.

F933H CENCNT: DEFW 0000H

In deze variabele houdt de "CIRCLE"-routine bij hoeveel punten geplot moeten worden om de eindhoek af te werken.

F935H CLINEF: DEFB 00H

Deze variabele dient als een vlag voor de "CIRCLE"-routine. Bit 0 wordt geset indien een lijn getrokken dient te worden vanaf de starthoek naar het middelpunt, bit 7 wordt geset als dit moet gebeuren vanaf de eindhoek.

F936H CNPNTS: DEFW 0000H

In deze variabele slaat de "CIRCLE"-routine op hoeveel punten een segment van vijfenveertig graden omvat.

F938H CPLOTF: DEFB 00H

Deze variabele bevat normaal nul. Zijn inhoud wordt gewijzigd door de "CIRCLE"-routine indien de eindhoek kleiner is dan de starthoek. Deze variabele wordt gebruikt om uit te maken of de pixels "binnen" of "buiten" die hoeken geset moeten worden.

F939H CPCNT: DEFW 0000H

In deze variabele slaat de "CIRCLE"-routine het aantal punten binnen het getekende segment van vijfenveertig graden op. In feite is het het Y-coördinaat.

F93BH CPCNT8: DEFW 0000H

In deze variabele zet de "CIRCLE"-routine het totaal aantal geplotte punten tot op de huidige positie.

F93DH CRCSUM: DEFW 0000H

Deze variabele wordt door de "CIRCLE"-routine gebruikt als teller bij het berekenen van het aantal punten.

F93FH CSTCNT: DEFW 0000H

Deze variabele wordt door de "CIRCLE"-routine gebruikt om het aantal punten binnen de starthoek in op te slaan.

F941H CSCLXY: DEFB 00H

Deze variabele dient als een vlag voor de "CIRCLE"-routine. Hij geeft aan in welke richting de elliptische afplatting moet gebeuren : 00H = Y-as, 01H = X-as.

F942H CSAVEA: DEFW 0000H

Deze variabele wordt door de standaardroutine SCANR gebruikt als tijdelijke opslagruimte.

F944H CSAVEM: DEFB 00H

Deze variabele wordt door de standaardroutine SCANR gebruikt als tijdelijke opslagruimte.

F945H CXOFF: DEFW 0000H

Deze variabele wordt door de "CIRCLE"-routine gebruikt als tijdelijke opslagruimte.

F947H CYOFF: DEFW 0000H

Deze variabele wordt door de "CIRCLE"-routine gebruikt als tijdelijke opslagruimte.

F949H LOHMSK: DEFB 00H

Deze variabele wordt door de "PAINT"-routine gebruikt om de meest linkse positie van een uitstulping naar links op te slaan.

F94AH LOHDIR: DEFB 00H

Deze variabele wordt door de "PAINT"-routine gebruikt om de nieuwe werkrichting, die door een uitstulping naar links wordt vereist, op te slaan.

F94BH LOHADR: DEFW 0000H

Deze variabele wordt door de "PAINT"-routine gebruikt om de meest linkse positie van een uitstulping naar links op te slaan.

F94DH LOHCNT: DEFW 0000H

In deze variabele zet de "PAINT"-routine de grootte van een uitstulping naar links.

F94FH SKPCNT: DEFW 0000H

Deze variabele wordt door de "PAINT"-routine gebruikt om er het aantal pixels in op te slaan, die overgeslagen mogen worden, zoals dat door de standaardroutine SCANR werd berekend.

F951H MOVCNT: DEFW 0000H

Deze variabele wordt door de "PAINT"-routine gebruikt om er het aantal bewegingen in op te slaan, dat door de standaardroutine SCANR werd berekend.

F953H PDIREC: DEFB 00H

Deze variabele wordt door de "PAINT"-routine gebruikt om er de richting in op te slaan waarin ze op dat ogenblik werkt : 40H - omlaag, C0H - omhoog, 00H - stoppen.

F954H LFPROG: DEFB 00H

Deze variabele bevat normaal nul. Zijn inhoud wordt gewijzigd door de "PAINT"-routine indien ze naar links heeft gewerkt.

F955H RTPROG: DEFB 00H

Deze variabele bevat normaal nul. Zijn inhoud wordt gewijzigd door de "PAINT"-routine indien ze naar rechts heeft gewerkt.

F956H MCLTAB: DEFW 0000H

Deze variabele bevat het adres van de tabel die de macro-ontleder moet gebruiken. De tabel voor het "DRAW"-statement staat op adres 5D83H, die voor het "PLAY"-statement op adres 752EH.

F958H MCLFLG: DEFB 00H

Deze variabele bevat nul indien de macro-ontleder voor het "DRAW"-statement wordt gebruikt. Hij verschilt van nul indien de macro-ontleder voor het "PLAY"-statement wordt gebruikt.

```
F959H  QUETAB: DEFB 00H   ; AQ bijzet-positie
F95AH          DEFB 00H   ; AQ ophaal-positie
F95BH          DEFB 00H   ; AQ terugzet-vlag
F95CH          DEFB 7FH   ; AQ grootte
F95DH          DEFW F975H ; AQ adres
F95FH          DEFB 00H   ; BQ bijzet-positie
F960H          DEFB 00H   ; BQ ophaal-positie
F961H          DEFB 00H   ; BQ terugzet-vlag
F962H          DEFB 7FH   ; BQ grootte
F963H          DEFW F9F5H ; BQ adres
F965H          DEFB 00H   ; CQ bijzet-positie
F966H          DEFB 00H   ; CQ ophaal-positie
F967H          DEFB 00H   ; CQ terugzet-vlag
F968H          DEFB 7FH   ; CQ grootte
F969H          DEFW FA75H ; CQ adres

F96BH          DEFB 00H   ; RQ bijzet-positie
F96CH          DEFB 00H   ; RQ ophaal-positie
F96DH          DEFB 00H   ; RQ terugzet-vlag
F96EH          DEFB 7FH   ; RQ grootte
F96FH          DEFW 0000H ; RQ adres
```

Deze vierentwintig variabelen vormen de controleblokken voor de drie muziekrijen (VOICAQ, VOICBQ en VOICCQ) en de RS232-rij. De drie controleblokken voor de muziek worden door de standaardroutine GICINI geïnitialiseerd en daarna bijgehouden door de interrupt-verwerker en de standaardroutine PUTQ. Het controleblok voor de RS232-uitgang wordt in de huidige MSX ROM niet gebruikt.

```
F971H  QUEBAK: DEFB 00H   ; AQ teruggezet karakter
               DEFB 00H   ; BQ teruggezet karakter
               DEFB 00H   ; CQ teruggezet karakter
               DEFB 00H   ; RQ teruggezet karakter
```

Deze vier variabelen worden gebruikt om een ongewenst karakter dat naar de bijbehorende muziekrij werd teruggestuurd, op te slaan. Hoewel deze voorziening in de MSX ROM werd ingebouwd, wordt ze momenteel niet gebruikt.

```
F975H  VOICAQ: DEFS 128   ; rij voor kanaal A
F9F5H  VOICBQ: DEFS 128   ; rij voor kanaal B
FA75H  VOICCQ: DEFS 128   ; rij voor kanaal C
FAF5H   RS2IQ: DEFS 64    ; rij voor RS232-kanaal
```

Deze vier buffers bevatten de drie muziekrijen en de RS232-rij. Deze laatste wordt niet gebruikt.

FB35H PRSCNT: DEFB ØØH

Deze variabele wordt door de "PLAY"-routine gebruikt om het aantal afgewerkte strings met operand te tellen. Nadat elk van de drie operands ontleed werd, wordt ook bit 7 geset, om te voorkomen dat de standaardroutine STRTMS herhaald wordt aangeroepen.

FB36H SAVSP: DEFW ØØØØH

Deze variabele wordt door de "PLAY"-routine gebruikt om er de inhoud van de Stack Pointer van de Z8Ø in op te slaan, vooraleer de controle overgedragen wordt aan de macro-ontleder. Op het einde van de ontleding, wordt de inhoud van deze variabele vergeleken met de Stack Pointer van de Z8Ø, om te controleren of er nog data op de stack stonden, die daar werden gezet na het voortijdig afbreken van de ontledingsroutine omdat de muziekrij vol was.

FB38H VOICEN: DEFB ØØH

Deze variabele bevat het nummer van het kanaal dat op dat ogenblik door de "PLAY"-routine wordt gebruikt. De waarden Ø, 1 en 2 komen overeen met de kanalen A, B en C van de Programmeerbare Geluidsgenerator.

FB39H SAVVOL: DEFW ØØØØH

Deze variabele wordt gebruikt door de routine die het "R"-commando van de "PLAY"-routine verwerkt, om tijdens een rust (amplitude nul) de vorige waarde van het volume in op te slaan.

FB3BH MCLLEN: DEFB ØØH

Deze variabele wordt door de macro-ontleder gebruikt om de lengte van de string, die op dat ogenblik wordt ontleed, in op te slaan.

FB3CH MCLPTR: DEFW ØØØØH

Deze variabele wordt door de macro-ontleder gebruikt om het adres van de string, die op dat ogenblik wordt ontleed, in op te slaan.

FB3EH QUEUEN: DEFB ØØH

Deze variabele wordt door de interruptverwerker gebruikt om het nummer van de muziekrij, die op dat ogenblik wordt verwerkt, in op te slaan. De waarden Ø, 1 en 2 komen overeen met de kanalen A, B en C van de Programmeerbare Geluidsgenerator.

FB3FH MUSICF: DEFB ØØH

Drie bits van deze variabele dienen als vlaggen. Ze worden door de standaardroutine STRTMS geset om aan de interruptverwerker te signaleren dat een bepaalde muziekrij verwerkt dient te worden. Bits Ø, 1 en 2 komen overeen met VOICAQ, VOICBQ en VOICCQ.

FB40H PLYCNT: DEFB 00H

Deze variabele wordt door de standaardroutine STRTMS gebruikt om er het aantal opeenvolgende "PLAY"-statements in op te slaan die op dat ogenblik in de muziekrijen staan. Zijn inhoud wordt onderzocht wanneer voor één uitvoering alle eindmarkeerders in de muziekrijen werden gelezen, om te bepalen of de muziekrijen opnieuw verwerkt dienen te worden.

```
FB41H    VCBA: DEFW 0000H ; teller van de tijdsduur
FB43H          DEFB 00H   ; lengte van de string
FB44H          DEFW 0000H ; adres van de string
FB46H          DEFW 0000H ; adres van data op de stack
FB48H          DEFB 00H   ; lengte van het muziekpakket
FB49H          DEFS 7     ; muziekpakket
FB50H          DEFB 04H   ; octaaf
FB51H          DEFB 04H   ; lengte
FB52H          DEFB 78H   ; tempo
FB53H          DEFB 88H   ; volume
FB54H          DEFW 00FFH ; periode van de envelope
FB56H          DEFS 16    ; ruimte voor data op de stack
```

Deze buffer van zevenendertig bytes bevat de parameters die de "PLAY"-routine nodig heeft om kanaal A te sturen.

FB66H VCBB: DEFS 37

Deze buffer bevat de parameters die de "PLAY"-routine nodig heeft om kanaal B te sturen. De structuur van de buffer is dezelfde als die van de buffer voor kanaal A.

FB8BH VCBC: DEFS 37

Deze buffer bevat de parameters die de "PLAY"-routine nodig heeft om kanaal C te sturen. De structuur van de buffer is dezelfde als die van de buffer voor kanaal A.

FBB0H ENSTOP: DEFB 00H

De inhoud van deze variabele bepaalt of de interruptverwerker een "warme start" zal uitvoeren, dit wil zeggen de Interpreter initialiseren, wanneer de toetsen "CODE", "GRPH", "CTRL" en "SHIFT" tegelijk worden ingedrukt. Bevat de variabele 00H, dan gebeurt dat niet; bevat hij iets anders, dan gebeurt het wel.

FBB1H BASROM: DEFB 00H

De inhoud van deze variabele bepaalt of de standaardroutines ISCNTC en INLIN reageren op het indrukken van de "CTRL/STOP"-toetsen. Bevat de variabele 00H, dan hebben de toetsen het gewone effekt; bevat hij iets anders, dan hebben ze geen effekt. Deze variabele dient om te voorkomen dat een BASIC programma in ROM, dat werd gevonden tijdens de initialisatie na het inschakelen, onderbroken wordt.

FBB2H LINTTB: DEFS 24

Elk van deze vierentwintig variabelen verschilt normaal van nul. Indien de inhoud van een regel op het scherm doorloopt op de volgende regel, wordt de overeenkomstige variabele nul gemaakt.

Het BIOS zorgt daarvoor, maar enkel de standaardroutine INLIN (de scherm-editor) maakt van deze variabelen gebruik om het onderscheid te maken tussen een logische regel en een regel op het scherm.

FBCAH FSTPOS: DEFW 0000H

Deze variabele bevat de coördinaten van de cursor, bij de start van de standaardroutine INLIN. Hij geeft aan, hoever terug gelezen moet worden, wanneer op het einde van de routine de tekst van het scherm wordt gehaald.

FBCCH CURSAV: DEFB 00H

Deze variabele wordt gebruikt om het karakter in op te slaan, dat op het scherm werd vervangen door de cursor.

FBCDH FNKSWI: DEFB 00H

Deze variabele wordt gebruikt door de standaardroutine CHSNS om te bepalen welke groep van functietoetsen op het scherm afgedrukt moet worden. Is de inhoud van de variabele 00H, dan is het de tekst van de laatste vijf (met "SHIFT"), is hij 01H dan is het de tekst van de eerste vijf (zonder "SHIFT").

FBCEH FNKFLG: DEFS 10

Elk van deze tien variabelen bevat normaal nul. De inhoud van een variabele wordt 01H gemaakt, als de overeenkomstige functietoets in werking werd gesteld door een "KEY (n) ON"-statement. Deze variabelen worden gebruikt door de interrupt-verwerker, om te bepalen of (enkel tijdens de loop van een programma) een tekst afgedrukt moet worden ofwel de bijbehorende plaats in TRPTBL aangepast dient te worden.

FBD8H ONGSBF: DEFB 00H

Deze variabele bevat normaal nul. De inhoud ervan wordt opgehoogd door de interruptverwerker, telkens wanneer een apparaat de voorwaarden vervult, die nodig zijn om een interrupt te produceren. De Verwerkingslus gebruikt deze variabele om te bepalen of er interrupts voorgekomen zijn, die nog afgewerkt moeten worden, zonder dat daarvoor TRPTBL doorzocht hoeft te worden.

FBD9H CLIKFL: DEFB 00H

Deze variabele wordt intern door de interruptverwerker gebruikt, om te voorkomen dat ongepaste klikjes zouden worden geproduceerd, wanneer het indrukken van één toets meerdere karakters doet afdrukken, zoals bijvoorbeeld functietoetsen.

FBDAH OLDKEY: DEFS 11

Deze buffer wordt door de interruptverwerker gebruikt om de vorige toestand van de matrix van het toetsenbord in op te slaan. Elk byte bevat een rij toetsen, te beginnen met rij 0.

FBE5H NEWKEY: DEFS 11

Deze buffer wordt door de interruptverwerker gebruikt om de huidige toestand van de matrix van het toetsenbord in op te slaan. Overgangen van toetsen worden opgespoord door de inhoud van deze buffer te vergelijken met de inhoud van OLDKEY. Daarna wordt OLDKEY aangepast aan de laatste toestand.

FBF0H KEYBUF: DEFS 40

Deze buffer bevat de karakters die de interrupt-verwerker uit de toets-indrukken decodeerde. Merk op dat deze buffer als een cirkelvormige rij is georganiseerd, die beheerd wordt door GETPNT en PUTPNT, en die daardoor geen vast startadres heeft.

FC18H LINWRK: DEFS 40

Deze buffer wordt door het BIOS gebruikt om een volledige regel van het scherm in op te slaan.

FC40H PATWRK: DEFS 8

Deze buffer wordt door het BIOS gebruikt om een pixelpatroon van 8 bij 8 pixels in op te slaan.

FC48H BOTTOM: DEFW 8000H

Deze variabele bevat het adres van het laagste adres in RAM dat door de Interpreter gebruikt wordt. Hij krijgt zijn waarde bij het inschakelen en die wordt nadien niet meer gewijzigd.

FC4AH HIMEM: DEFW F380H

Deze variabele bevat het adres van het byte nā het hoogste adres in RAM dat nog door de Interpreter gebruikt wordt. Hij krijgt zijn waarde bij het inschakelen en wordt nadien enkel nog gewijzigd door het "CLEAR"-statement.

De volgende zesentwintig variabelen van elk drie bytes bevatten de momentele toestand van alle toestellen die een interrupt kunnen produceren. Het eerste byte van de drie bevat de status van het toestel (bit 0 - in werking, bit 1 - stop, bit 2 - "event" in werking).

Het wordt door de interrupt-verwerker, de interrupt-routine van de Verwerkingslus en de "DEVICE ON/OFF/STOP"-routine en de "RETURN"-routine aangepast. De overige twee bytes van elke groep worden ingevuld door de "ON DEVICE GOSUB"-routine : ze bevatten het adres van de programmaregel die moet worden uitgevoerd indien dat bepaalde toestel een interrupt veroorzaakt.

(zie tabel op de volgende bladzijde)

```
FC4CH  TRPTBL: DEFS 3   ; KEY 1
FC4FH          DEFS 3   ; KEY 2
FC52H          DEFS 3   ; KEY 3
FC55H          DEFS 3   ; KEY 4
FC58H          DEFS 3   ; KEY 5
FC5BH          DEFS 3   ; KEY 6
FC5EH          DEFS 3   ; KEY 7
FC61H          DEFS 3   ; KEY 8
FC64H          DEFS 3   ; KEY 9
FC67H          DEFS 3   ; KEY 10
FC6AH          DEFS 3   ; STOP
FC6DH          DEFS 3   ; SPRITE
FC70H          DEFS 3   ; STRIG 0
FC73H          DEFS 3   ; STRIG 1
FC76H          DEFS 3   ; STRIG 2
FC79H          DEFS 3   ; STRIG 3
FC7CH          DEFS 3   ; STRIG 4
FC7FH          DEFS 3   ; INTERVAL
FC82H          DEFS 3   ; ongebruikt
FC85H          DEFS 3   ; ongebruikt
FC88H          DEFS 3   ; ongebruikt
FC8BH          DEFS 3   ; ongebruikt
FC8EH          DEFS 3   ; ongebruikt
FC91H          DEFS 3   ; ongebruikt
FC94H          DEFS 3   ; ongebruikt
FC97H          DEFS 3   ; ongebruikt
```

FC9AH RTYCNT: DEFB 00H

Deze variabele wordt momenteel in de MSX ROM niet gebruikt.

FC9BH INTFLG: DEFB 00H

Deze variabele bevat normaal nul. De interruptverwerker zet de inhoud op 03H of 04H indien hij de "CTRL/STOP" of de "STOP"-toets detecteert.

FC9CH PADY: ~~LEFB 00H~~ DEFB 00H

Deze variabele ~~be~~ bevat het Y-coördinaat van het laatste punt dat door een electronisch tekenbord werd ~~gedetecteerd~~ gedetecteerd.

FC9DH PADX: DEFB 00H

Deze variabele bevat het X-coördinaat van het laatste punt dat door een electronisch tekenbord werd gedetecteerd.

FC9EH JIFFY: DEFW 0000H

Deze variabele wordt continu opgehoogd door de interrupt-verwerker. Zijn inhoud kan uitgelezen worden of gewijzigd worden door respectievelijk de "TIME"-functie of het "TIME"-statement.

FCA0H INTVAL: DEFW 0000H

Deze variabele bevat de duur van het interval, dat door de "ON INTERVAL"-routine werd ingesteld.

FCA2H INTCNT: DEFW 0000H

Deze variabele wordt continu met één verlaagd door de interruptverwerker. Wanneer hij op nul komt, krijgt hij de waarde die in INTVAL staat en wordt, indien dit van toepassing is, een interrupt geproduceerd. Merk op dat het aftellen van deze variabele altijd doorgaat, los van het feit of er een "INTERVAL ON"-statement in werking is of niet.

FCA4H LOWLIM: DEFB 31H

Deze variabele wordt gebruikt om er de minimaal toegelaten duur van het startbit in op te slaan, die door de standaardroutine TAPION werd bepaald.

FCA5H WINWID: DEFB 22H

Deze variabele wordt gebruikt om er de tijdsduur in op te slaan die het verschil zal uitmaken tussen een LAAG- en een HOOG-cyclus. Die duur wordt door de standaardroutine TAPION bepaald.

FCA6H GRPHED: DEFB 00H

Deze variabele bevat normaal nul. De standaardroutine CNVCHR wijzigt dit in 01H wanneer een grafische header-code wordt gelezen.

FCA7H ESCCNT: DEFB 00H

Deze variabele wordt gebruikt door de routines die voor de standaardroutine CHPUT de "ESC"-groepen verwerken, om de parameters van de "ESC"-code te tellen.

FCA8H INSFLG: DEFB 00H

Deze variabele bevat normaal nul. De standaardroutine INLIN wijzigt dit in FFH wanneer de "INSERT"-mode in werking is, dat wil zeggen wanneer karakters in bestaande tekst tussengevoegd kunnen worden.

FCA9H CSRSR: DEFB 00H

Indien deze variabele nul bevat, wordt de cursor enkel op het scherm gezet zolang de standaardroutine CHGET wacht totdat een karakter wordt ingetypt. Bevat de variabele iets anders dan nul, dan wordt de cursor, via de standaardroutine CHPUT, permanent op het scherm gezet.

FCAAH CSTYLE: DEFB 00H

De inhoud van deze variabele bepaalt het uitzicht van de cursor. Bevat de variabele nul, dan is de cursor blokvormig. In het andere geval is de cursor 8 pixels breed en 4 pixels hoog.

FCABH CAPST: DEFB 00H

In deze variabele houdt de interrupt-verwerker bij of de hoofdletters uitgeschakeld of ingeschakeld zijn door er respectievelijk nul of een andere waarde in op te slaan.

FCACH KANAST: DEFB 00H

Bij Japanse machines geeft deze variabele aan of de Kana-stand al dan niet ingesteld staat. Bij Europese machines geeft hij de toestand van de DODE toets weer.

FCADH KANAMD: DEFB 00H

Enkel bij Japanse machines bevat deze variabele informatie over de toetsenbord-mode.

FCAEH FLBMEM: DEFB 00H

De inhoud van deze variabele wordt bepaald door de routines die fouten verwerken in verband met In/Out-bewerkingen. Hij wordt verder niet gebruikt.

FCAFH SCRMOD: DEFB 00H

Deze variabele bevat informatie over de huidige schermmode. De waarden 0 tot 3 stellen respectievelijk de 40x24 tekstmode, de 32x24 tekstmode, de grafische mode en de veelkleurenmode voor.

FCB0H OLDSCR: DEFB 00H

Deze variabele geeft aan in welke tekstmode het scherm het laatst werd gezet.

FCB1H CASPRV: DEFB 00H

In deze variabele wordt een karakter opgeslagen dat naar een In/Out-buffer werd teruggestuurd door de terugzetfunctie van de cassetteroutine.

FCB2H BRDATR: DEFB 00H

Deze variabele bevat de grenskleur voor de "PAINT"-routine. Hij krijgt zijn waarde van de standaardroutine PNTINI. Die waarde wordt gebruikt door de standaardroutines SCANR en SCANL.

FCB3H GXPOS: DEFW 0000H

Deze variabele wordt gebruikt voor de tijdelijke opslag van een grafisch X-coördinaat.

FCB5H GYPOS: DEFW 0000H

Deze variabele wordt gebruikt voor de tijdelijke opslag van een grafisch Y-coördinaat.

FCB7H GRPACX: DEFW 0000H

Deze variabele bevat het huidige X-coördinaat, ten behoeve van de standaardroutine GRPPRT.

FCB9H GRPACY: DEFW 0000H

Deze variabele bevat het huidige Y-coördinaat, ten behoeve van de standaardroutine GRPPRT.

FCBBH DRWFLG: DEFB 00H

Bits 6 en 7 van deze variabele worden geset door de "N"- en "B"-commando's van het "DRAW"-statement, om de bijbehorende mode in werking te stellen.

FCBDH DRWANG: DEFB 00H

Deze variabele wordt gebruikt door het "A"-commando van het "DRAW"-statement, om de gebruikte hoek in op te slaan.

FCBEH RUNBNF: DEFB 00H

Deze variabele bevat normaal nul. Zijn inhoud wordt gewijzigd door de "BLOAD"-routine indien de "R"-parameter voor automatisch opstarten werd gespecifieerd.

FCBFH SAVENT: DEFW 0000H

Deze variabele bevat het toegangsadres van een "BSAVE"- en "BLOAD"-statement.

```
FCC1H  EXPTBL: DEFB 00H    ; Primair Slot 0
FCC2H          DEFB 00H    ; Primair Slot 1
FCC3H          DEFB 00H    ; Primair Slot 2
FCC4H          DEFB 00H    ; Primair Slot 3
```

Deze vier variabelen bevatten normaal nul. Indien tijdens de initialisatie, bij het doorzoeken van het RAM, blijkt dat een Primair Slot werd uitgebreid, dan wordt de inhoud van de overeenkomstige variabele 80H.

```
FCC5H  SLTTBL: DEFB 00H    ; Primair Slot 0
FCC6H          DEFB 00H    ; Primair Slot 1
FCC7H          DEFB 00H    ; Primair Slot 2
FCC8H          DEFB 00H    ; Primair Slot 3
```

Deze vier variabelen vormen een kopie van de inhoud van de vier mogelijke Secundaire Slot-registers. De inhoud van elke variabele is dan slechts van belang, wanneer uit EXPTBL blijkt dat het overeenkomstige Primaire Slot een uitbreiding bevat.

```
FCC9H  SLTATR: DEFS 4      ; PS0, SS0
FCCDH          DEFS 4      ; PS0, SS1
FCD1H          DEFS 4      ; PS0, SS2
FCD5H          DEFS 4      ; PS0, SS3
FCD9H          DEFS 4      ; PS1, SS0
FCDDH          DEFS 4      ; PS1, SS1
FCE1H          DEFS 4      ; PS1, SS2
FCE5H          DEFS 4      ; PS1, SS3
FCE9H          DEFS 4      ; PS2, SS0
FCEDH          DEFS 4      ; PS2, SS1
FCF1H          DEFS 4      ; PS2, SS2
FCF5H          DEFS 4      ; PS2, SS3
FCF9H          DEFS 4      ; PS3, SS0
FCFDH          DEFS 4      ; PS3, SS1
FD01H          DEFS 4      ; PS3, SS2
FD05H          DEFS 4      ; PS3, SS3
```

Deze vierenzestig variabelen bevatten de attributen van de uitbreidings-ROMs die gevonden werden tijdens de initialisatie, bij het zoeken naar ROM. De karakteristieken van elke ROM van 16Kb worden in één byte gecodeerd, zodat vier bytes in totaal nodig zijn. De codering gebeurt als volgt :

bit 7 geset - deze ROM bevat een BASIC programma
bit 6 geset - deze ROM behandelt een bepaald apparaat
bit 5 geset - deze ROM verwerkt bepaalde statements

Merk op dat de bytes die overeenkomen met bladzijde 0 (0000H tot 3FFFH) en bladzijde 3 (C000H tot FFFFH) altijd nul zijn, omdat enkel bladzijde 1 (4000H tot 7FFFH) en bladzijde 2 (8000H tot BFFFH) worden onderzocht. In MSX geldt de afspraak dat uitbreidings-ROMs die machinetaal bevatten, op bladzijde 1 gezet worden, en ROMs die BASIC programma's bevatten, op bladzijde 2.

FD09H SLTWRK: DEFS 128

Deze buffer voorziet in een plaatselijk werkgebied van twee bytes voor elk van de vierenzestig mogelijke uitbreidings-ROMs.

FD89H PROCNM: DEFS 16

Deze buffer wordt gebruikt om de naam te bevatten van een apparaat of een statement, dat door een uitbreidings-ROM onderzocht dient te worden.

FD99H DEVICE: DEFB 00H

Deze variabele wordt gebruikt om de code van een randapparaat, tussen 0 en 3, aan een uitbreidings-ROM door te geven.

DE "HOOKS"

Het stuk van het Werkgebied tussen FDA9H en FFC9H bevat honderd en twaalf "hooks". Bij het inschakelen van de machine worden die allemaal met vijf "RET"-opcodes gevuld (C9H). Deze "hooks" worden opgeroepen vanuit strategische plaatsen in het BIOS of de Interpreter, zodanig dat de ROM uitgebreid kan worden, meer bepaald zodanig dat Disk BASIC toegevoegd kan worden. In elke "hook" is voldoende ruimte vrij voor een "verre" "CALL" naar elk Slot, op de volgende manier :

```
RST 30H
DEFB Slot-identificator
DEFW adres voor de "CALL"
RET
```

De "hooks" worden op de volgende bladzijden gegeven, samen met het adres van waar ze worden aangeroepen en een korte uitleg over hun functie.

Het stuk van het Werkgebied tussen FFCAH en FFFFH wordt niet gebruikt.

```
HOOK-ADRES      NAAM:  LENGTE   AANGEROEPEN ROUTINE
-----------------------------------------------------------------
FD9AH           HKEYI: DEFS 5 ; 0C4AH Interrupt verwerker
FD9FH           HTIMI: DEFS 5 ; 0C53H Interrupt verwerker
FDA4H           HCHPU: DEFS 5 ; 08C0H De standaardroutine CHPUT
FDA9H           HDSPC: DEFS 5 ; 09E6H Zet cursor op scherm
FDAEH           HERAC: DEFS 5 ; 0A33H Haal cursor van scherm
FDB3H           HDSPF: DEFS 5 ; 0B2BH De standaardroutine DSPFNK
FDB8H           HERAF: DEFS 5 ; 0B15H De standaardroutine ERAFNK
FDBDH           HTOTE: DEFS 5 ; 0842H De standaardroutine TOTEXT
FDC2H           HCHGE: DEFS 5 ; 10CEH De standaardroutine CHGET
FDC7H           HINIP: DEFS 5 ; 071EH Copieer karakterset naar VDP
FDCCH           HKEYC: DEFS 5 ; 1025H Toetsenbord decoder
FDD1H           HKEYA: DEFS 5 ; 0F10H Toetsenbord decoder
FDD6H            HNMI: DEFS 5 ; 1398H De standaardroutine NMI
FDDBH           HPINL: DEFS 5 ; 23BFH De standaardroutine PINLIN
FDE0H           HQINL: DEFS 5 ; 23CCH De standaardroutine QINLIN
FDE5H           HINLI: DEFS 5 ; 23D5H De standaardroutine INLIN
FDEAH           HONGO: DEFS 5 ; 7810H "ON DEVICE GOSUB"
FDEFH           HDSKO: DEFS 5 ; 7C16H "DSKO$"
FDF4H           HSETS: DEFS 5 ; 7C1BH "SET"
FDF9H           HNAME: DEFS 5 ; 7C20H "NAME"
FDFEH           HKILL: DEFS 5 ; 7C25H "KILL"
FE03H            HIPL: DEFS 5 ; 7C2AH "IPL"
FE08H           HCOPY: DEFS 5 ; 7C2FH "COPY"
FE0DH            HCMD: DEFS 5 ; 7C34H "CMD"
FE12H           HDSKF: DEFS 5 ; 7C39H "DSKF"
FE17H           HDSKI: DEFS 5 ; 7C3EH "DSKI$"
FE1CH           HATTR: DEFS 5 ; 7C43H "ATTR$"
FE21H           HLSET: DEFS 5 ; 7C48H "LSET"
FE26H           HRSET: DEFS 5 ; 7C4DH "RSET"
FE2BH           HFIEL: DEFS 5 ; 7C52H "FIELD"
FE30H           HMKI$: DEFS 5 ; 7C57H "MKI$"
FE35H           HMKS$: DEFS 5 ; 7C5CH "MKS$"
FE3AH           HMKD$: DEFS 5 ; 7C61H "MKD$"
FE3FH            HCVI: DEFS 5 ; 7C66H "CVI"
FE44H            HCVS: DEFS 5 ; 7C6BH "CVS"
FE49H            HCVD: DEFS 5 ; 7C70H "CVD"
FE4EH           HGETP: DEFS 5 ; 6A93H Localiseer controleblok
FE53H           HSETF: DEFS 5 ; 6AB3H Localiseer controleblok
FE58H           HNOFO: DEFS 5 ; 6AF6H "OPEN"
FE5DH           HNULO: DEFS 5 ; 6B0FH "OPEN"
FE62H           HNTFL: DEFS 5 ; 6B3BH Sluit I/O buffer 0
FE67H           HMERG: DEFS 5 ; 6B63H "MERGE / LOAD"
FE6CH           HSAVE: DEFS 5 ; 6BA6H "SAVE"
FE71H           HBINS: DEFS 5 ; 6BCEH "SAVE"
FE76H           HBINL: DEFS 5 ; 6BD4H "MERGE / LOAD"
FE7BH           HFILE: DEFS 5 ; 6C2FH "FILES"
FE80H           HDGET: DEFS 5 ; 6C3BH "GET / PUT"
FE85H           HFILO: DEFS 5 ; 6C51H Sequentiële output
FE8AH           HINDS: DEFS 5 ; 6C79H Sequentiële input
FE8FH           HRSLF: DEFS 5 ; 6CD8H "INPUT$"
FE94H           HSAVD: DEFS 5 ; 6D03H "LOC", 6D14H "LOF",
                              ; 6D25H  "EOF", 6D39H "FPOS"
FE99H            HLOC: DEFS 5 ; 6D0FH "LOC"
FE9EH            HLOF: DEFS 5 ; 6D20H "LOF"
FEA3H            HEOF: DEFS 5 ; 6D33H "EOF"
FEA8H           HFPOS: DEFS 5 ; 6D43H "FPOS"
FEADH           HBAKU: DEFS 5 ; 6E36H "LINE INPUT#"
FEB2H           HPARD: DEFS 5 ; 6F15H Onderzoek naam randapparaat
```

```
HOOK-ADRES       NAAM: LENGTE   AANGEROEPEN ROUTINE
----------------------------------------------------------------
FEB7H            HNODE: DEFS 5 ; 6F33H Onderzoek naam randapparaat
FEBCH            HPOSD: DEFS 5 ; 6F37H Onderzoek naam randapparaat
FEC1H            HDEVN: DEFS 5 ; Deze HOOK wordt niet gebruikt
FEC6H            HGEND: DEFS 5 ; 6F8FH De dispatcher
FECBH            HRUNC: DEFS 5 ; 629AH Run-clear
FED0H            HCLEA: DEFS 5 ; 62A1H Run-clear
FED5H            HLOPD: DEFS 5 ; 62AFH Run-clear
FEDAH            HSTKE: DEFS 5 ; 62F0H Initialiseer de Z-80 Stack
FEDFH            HISFL: DEFS 5 ; 145FH De standaardroutine ISFLIO
FEE4H            HOUTD: DEFS 5 ; 1B46H De standaardroutine OUTDO
FEE9H            HCRDO: DEFS 5 ; 7328H CR,LF naar OUTDO
FEEEH            HDSKC: DEFS 5 ; 7374H Inlezen van een regel tekst
FEF3H            HDOGR: DEFS 5 ; 593CH Lijn tekenen
FEF8H            HPRGE: DEFS 5 ; 4039H Eind van programma
FEFDH            HERRP: DEFS 5 ; 40DCH Foutverwerker
FF02H            HERRF: DEFS 5 ; 40FDH Foutverwerker
FF07H            HREAD: DEFS 5 ; 4128H "Ok" punt van de Hoofdlus
FF0CH            HMAIN: DEFS 5 ; 4134H De Hoofdlus
FF11H            HDIRD: DEFS 5 ; 41A8H Verwerken van direct commando
FF16H            HFINI: DEFS 5 ; 4237H Eind van Hoofdlus
FF1BH            HFINE: DEFS 5 ; 4247H Eind van Hoofdlus
FF20H            HCRUN: DEFS 5 ; 42B9H Coderen
FF25H            HCRUS: DEFS 5 ; 4353H Coderen
FF2AH            HISRE: DEFS 5 ; 437CH Coderen
FF2FH            HNTFN: DEFS 5 ; 43A4H Coderen
FF34H            HNOTR: DEFS 5 ; 44EBH Coderen
FF39H            HSNGF: DEFS 5 ; 45D1H "FOR"
FF3EH            HNEWS: DEFS 5 ; 4601H Nieuw statement verwerkingslus
FF43H            HGONE: DEFS 5 ; 4646H Uitvoeringspunt verwerkingslus
FF48H            HCHRG: DEFS 5 ; 4666H De standaardroutine CHRGTR
FF4DH            HRETU: DEFS 5 ; 4821H "RETURN"
FF52H            HPRTF: DEFS 5 ; 4A5EH "PRINT"
FF57H            HCOMP: DEFS 5 ; 4A94H "PRINT"
FF5CH            HFINP: DEFS 5 ; 4AFFH "PRINT"
FF61H            HTRMN: DEFS 5 ; 4B4DH Fout bij "READ / INPUT"
FF66H            HFRME: DEFS 5 ; 4C6DH Expression Evaluator
FF6BH            HNTPL: DEFS 5 ; 4CA6H Expression Evaluator
FF70H            HEVAL: DEFS 5 ; 4DD9H Factor Evaluator
FF75H            HOKNO: DEFS 5 ; 4F2CH Factor Evaluator
FF7AH            HFING: DEFS 5 ; 4F3EH Factor Evaluator
FF7FH            HISMI: DEFS 5 ; 51C3H Uitvoeringspunt verwerkingslus
FF84H            HWIDT: DEFS 5 ; 51CCH "WIDTH"
FF89H            HLIST: DEFS 5 ; 522EH "LIST"
FF8EH            HBUFL: DEFS 5 ; 532DH Decoderen
FF93H            HFRQI: DEFS 5 ; 543FH Omzetten naar geheel getal
FF98H            HSCNE: DEFS 5 ; 5514H Regelnummer naar adrespointer
FF9DH            HFRET: DEFS 5 ; 67EEH Verwijderen van signalement
FFA2H            HPTRG: DEFS 5 ; 5EA9H De variabelen zoekroutine
FFA7H            HPHYD: DEFS 5 ; 148AH De standaardroutine PHYDIO
FFACH            HFORM: DEFS 5 ; 148EH De standaardroutine FORMAT
FFB1H            HERRO: DEFS 5 ; 406FH De foutverwerker
FFB6H            HLPTO: DEFS 5 ; 085DH De standaardroutine LPTOUT
FFBBH            HLPTS: DEFS 5 ; 0884H De standaardroutine LPTSTT
FFC0H            HSCRE: DEFS 5 ; 79CCH "SCREEN"
FFC5H            HPLAY: DEFS 5 ; 73E5H "PLAY"
```

HOOFDSTUK 7

MACHINETAAL-PROGRAMMA'S

Dit hoofdstuk bevat een aantal programma's in machinetaal, waarin wordt aangetoond hoe het MSX-systeem nuttig gebruikt kan worden. De programma's werden met behulp van de ZEN Assembler geschreven. Ze zijn ontworpen om in BASIC programma's gebruikt te worden. De programma's kunnen, indien nodig, als een reeks hexadecimale codes ingevoerd worden met behulp van het korte BASIC programma hieronder. Het is wijs, om de geproduceerde code eerst op cassette te zetten, vooraleer ze te laten uitvoeren.

```
10 CLEAR 200,&HE000
20 ADDR - &HE000
30 PRINT RIGHT$("000"+HEX$(ADDR),4);
40 INPUT D$
50 POKE ADDR,VAL("&H"+D$)
60 ADDR=ADDR+1
70 GOTO 30
```

Alle programma's hebben als beginadres E000H, en worden op datzelfde adres aangeroepen. Tenzij dit anders wordt vermeld, hoeven geen parameters aan de programma's te worden doorgegeven. Daardoor kan een programma gestart worden met het statement :

```
DEFUSR=&HE000: ?USR(0)
```

De matrix van het toetsenbord

Dit programma toont de matrix van het toetsenbord op het scherm. Zo kan u direct zien welke toets werd ingedrukt. U kunt het programma stoppen door de "CTRL" en "STOP"-toetsen tegelijk in te drukken. Onder bepaalde omstandigheden kunnen onjuiste resultaten op het scherm getoond worden, indien meer dan drie of vier toetsen tegelijk worden ingedrukt. Dit is karakteristiek voor alle toetsenborden van het matrix-type.

Matrix van het toetsenbord uitlezen

```
                          ORG       0E000H
                          LOAD      0E000H

                    ;****************************
                    ;*  BIOS STANDAARDROUTINES  *
                    ;****************************

                      INITXT:EQU     006CH
                      CHPUT: EQU     00A2H
                      SNSMAT:EQU     0141H
                      BREAKX:EQU     00B7H

                    ;****************************
                    ;* VARIABELEN IN WERKGEBIED *
                    ;****************************

                      INTFLG:EQU     0FC9H

                    ;****************************
                    ;* CONTROLE-KARAKTERS       *
                    ;****************************

                          LF:     EQU      10
                          HOME:   EQU      11
                          CR:     EQU      13

ADRES  HEXcode LABEL   MNEMONICS           COMMENTAAR

E000   CD6C00  MATRIX: CALL INITXT         ; SCREEN 0
E003   3E0B    MX1:    LD   A,HOME         ;
E005   CDA200          CALL CHPUT          ; cursor op "HOME"
E008   AF              XOR  A              ; A=rij v toetsenbord
E009   F5      MX2:    PUSH AF             ;
E00A   CD4101          CALL SNSMAT         ; tast rij af
E00D   0608            LD   B,8            ; acht kolommen
E00F   07      MX3:    RLCA                ; kies 1 kolom
E010   F5              PUSH AF             ;
E011   E601            AND  1              ;
E013   C630            ADD  A,"0"          ; resultaat
E015   CDA200          CALL CHPUT          ; toon kolom
E018   F1              POP  AF             ;
E019   10F4            DJNZ MX3            ;
E01B   3E0D            LD   A,CR           ; volgende regel
E01D   CDA200          CALL CHPUT          ;
E020   3E0A            LD   A,LF           ;
E022   CDA200          CALL CHPUT          ;
E025   F1              POP  AF             ; A=rij v toetsenbord
E026   3C              INC  A              ; volgende rij
E027   FE0B            CP   11             ; klaar ?
E029   20DE            JR   NZ,MX2         ;
E02B   CDB700          CALL BREAKX         ; CTRL/STOP ?
E02E   30D3            JR   NC,MX1         ; ga verder
E030   AF              XOR  A              ;
E031   329BFC          LD   (INTFLG),A     ; hef "STOP" op
E034   C9              RET                 ; naar BASIC

                       END
```

Tekst in grafische mode over 40 kolommen

Dit programma laat toe om op het grafisch scherm, veertig karakters per regel af te drukken. De string die moet worden afgedrukt, wordt als parameter van de "USR"-functie meegegeven, bijvoorbeeld in het statement : A$ = USR ("een string"). Het is niet nodig om vooraf een "GRP"-file te openen. Het programma vraagt alleen maar dat het scherm in de correcte mode staat. De kern van het programma is qua functie gelijk aan de standaard-routine GRPPRT, met dit verschil dat enkel de eerste 6 pixelkolommen van een bepaald karakterpatroon op het scherm worden gezet in plaats van alle acht. Zoals bij GRPPRT wordt ook door deze routine het karakter geschreven op de huidige grafische positie. Het enige controlekarakter dat wordt herkend is "CR" (code 13), dat wordt uitgevoerd als de combinatie "CR"/"LF". In tegenstelling met de standaardroutine GRPPRT zullen karakters die op negatieve coördinaten worden afgedrukt, maar nog voor een deel op het scherm staan, op de correcte manier worden afgedrukt. Zoals het in dit boek staat, voert het programma automatisch een "LF"-code uit nadat het X-coördinaat 239 werd bereikt, waardoor dus precies veertig karakters op een regel worden gezet. Dit kan, indien nodig, gewijzigd worden door de constante in de subroutine RMDCOL aan te passen, zodanig dat de volle breedte van het scherm gebruikt kan worden.

Grafische mode : tekst 40 kolommen

```
                         ORG     E000H
                         LOAD    E000H

                   ;****************************
                   ;* BIOS STANDAARDROUTINES   *
                   ;****************************

                      RSLT:   EQU     000CH
                      CNVCHR: EQU     00ABH
                      MAPXYC: EQU     0111H
                      SETC:   EQU     0120H

                   ;****************************
                   ;* VARIABELEN IN WERKGEBIED *
                   ;****************************

                      FORCLR: EQU     0F3E9H
                      ATRBYT: EQU     0F3F2H
                      CGPNT:  EQU     0F91FH
                      PATWRK: EQU     0FC40H
                      SCRMOD: EQU     0FCAFH
                      GRPACX: EQU     0FCB7H
                      GRPACY: EQU     0FCB9H

                   ;****************************
                   ;* CONTROLE-KARAKTERS       *
                   ;****************************

                         CR:     EQU     13

ADRES  HEXcode LABEL   MNEMONICS          COMMENTAAR

E000   FE03    GFORTY: CP   3           ; is string ?
E002   C0              RET  NZ          ;
E003   3AAFFC          LD   A,(SCRMOD)  ; scherm-mode
E006   FE02            CP   2           ; grafische mode ?
E008   C0              RET  NZ          ;
E009   EB              EX   DE,HL       ; HL-> signalement
E00A   46              LD   B,(HL)      ; B=lengte v string
E00B   23              INC  HL          ;
E00C   5E              LD   E,(HL)      ; LSB van het adres
E00D   23              INC  HL          ;
E00E   56              LD   D,(HL)      ; DE-> string
E00F   04              INC  B           ;
E010   05      GF2:    DEC  B           ; klaar ?
E011   C8              RET  Z           ;
E012   1A              LD   A,(DE)      ; A=karakter uit string
E013   CD19E0          CALL GPRINT      ; druk af
E016   13              INC  D           ;
E017   18F7            JR   GF2         ; volgende karakter
```

```
ADRES  HEXcode  LABEL    MNEMONICS         COMMENTAAR

E019   F5       GPRINT:  PUSH AF           ;
E01A   C5                PUSH BC           ;
E01B   D5                PUSH DE           ;
E01C   E5                PUSH HL           ;
E01D   FDE5              PUSH IY           ;
E01F   ED4BB7FC          LD   BC,(GRPACX); BC= X-coördinaat
E023   ED5BB9FC          LD   DE,(GRPACY); DE= Y-coördinaat
E027   CD39E0            CALL GDC          ; decodeer karakter
E02A   ED43B7FC          LD   (GRPACX),BC; nieuw X-coördinaat
E02E   ED53B9FC          LD   (GRPACY),DE; nieuw Y-coördinaat
E032   FDE1              POP  IY           ;
E034   E1                POP  HL           ;
E035   D1                POP  DE           ;
E036   C1                POP  BC           ;
E037   F1                POP  AF           ;
E038   C9                RET               ;

E039   CDAB00   GDC:     CALL CNVCHR       ; grafisch karakter ?
E03C   D0                RET  NC           ; NC = gr. header
E03D   2007              JR   NZ,GD2       ; NZ = omgezet kar.
E03F   FE0D              CP   CR           ; carriage return ?
E041   2873              JR   Z,GCRLF      ;
E043   FE20              CP   20H          ; andere controlecode ?
E045   D8                RET  C            ; negeren !
E046   6F       GD2:     LD   L,A          ;
E047   2600              LD   H,0          ; HL=karaktercode
E049   29                ADD  HL,HL        ;
E04A   29                ADD  HL,HL        ;
E04B   29                ADD  HL,HL        ; HL= kar.code * 8
E04C   C5                PUSH BC           ; X-coördinaat
E04D   D5                PUSH DE           ; Y-coördinaat
E04E   ED5B20F9          LD   DE,(CGPNT+1); karakterset
E052   19                ADD  HL,DE        ; HL-> patroon
E053   1140FC            LD   DE,PATWRK    ; DE-> buffer
E056   0608              LD   B,8          ; patroon=8 bytes
E058   C5       GD3:     PUSH BC           ;
E059   D5                PUSH DE           ;
E05A   3A1FF9            LD   A,(CGPNT)    ; slot-identificator
E05D   CD0C00            CALL RDSLT        ; haal patroon op
E060   FB                EI                ;
E061   D1                POP  DE           ;
E062   C1                POP  BC           ;
E063   12                LD   (DE),A       ; zet in buffer
E064   13                INC  DE           ;
E065   23                INC  HL           ;
E066   10F0              DJNZ GD3          ; volgende
E068   D1                POP  DE           ; Y-coördinaat
E069   C1                POP  BC           ; X-coördinaat
E06A   3AE9F3            LD   A,(FORCLR)   ; huidige kleur
E06D   32F2F3            LD   (ATRBYT),A   ; vul inktkleur in
E070   FD2140FC          LD   IY,PATWRK    ; IY-> patronen
E074   D5                PUSH DE           ;
E075   2608              LD   H,8          ; max. pixelrijen
E077   CB7A     GD4:     BIT  7,D          ; positie Y-coörd. ?
E079   202A              JR   NZ,GD8       ;
E07B   CDBFE0            CALL BMDROW       ; laagste pixelrij ?
E07E   382B              JR   C,GD9        ; C = Y te groot
E080   C5                PUSH BC           ;
E081   2E06              LD   L,6          ; max. pixelkolommen
```

```
ADRES  HEXcode LABEL   MNEMONICS          COMMENTAAR

E083   FD7E00          LD    A,(IY+0)     ; A=rij v.h. patroon
E086   CB78    GD5:    BIT   7,B          ; positie X-coördinaat
E088   2015            JR    NZ,GD6       ;
E08A   CDC8E0          CALL  RMDCOL       ; meest rechtse kolom ?
E08D   3815            JR    C,GD7        ; C = X te groot
E08F   CB7F            BIT   7,A          ; bit uit patroon
E091   280C            JR    Z,GD6        ; Z = "0"-pixel
E093   F5              PUSH  AF           ;
E094   D5              PUSH  DE           ;
E095   E5              PUSH  HL           ;
E096   CD1101          CALL  MAPXYC       ; vind coördinaten
E099   CD2001          CALL  SETC         ; set de pixel
E09C   E1              POP   HL           ;
E09D   D1              POP   DE           ;
E09E   F1              POP   AF           ;
E09F   07      GD6:    RLCA               ; verschuif patroon
E0A0   03              INC   BC           ; X=X+1
E0A1   2D              DEC   L            ; kolommen afgewerkt ?
E0A2   20E2            JR    NZ,GD5       ;
E0A4   C1      GD7:    POP   BC           ; oorspronk. X-coörd.
E0A5   FD23    GD8:    INC   IY           ; vlgde byte uit patroon
E0A7   13              INC   DE           ; Y=Y+1
E0A8   25              DEC   H            ; rijen afgewerkt ?
E0A9   20CC            JR    NZ,GD4       ;
E0AB   D1      GD9:    POP   DE           ; oorspronk. Y-coörd.
E0AC   210600          LD    HL,6         ; step-waarde
E0AF   09              ADD   HL,BC        ; X=X+6
E0B0   44              LD    B,H          ;
E0B1   4D              LD    C,L          ; BC= nieuwe X-coörd.
E0B2   CDC8E0          CALL  RMDCOL       ; meest rechtse kolom ?
E0B5   D0              RET   NC           ;

E0B6   010000  GCRLF:  LD    BC,0         ; X=0
E0B9   210800          LD    HL,8         ;
E0BC   19              ADD   HL,DE        ;
E0BD   EB              EX    DE,HL        ; Y=Y+8
E0BE   C9              RET                ;

E0BF   E5      BMDROW: PUSH  HL           ;
E0C0   21BF00          LD    HL,191       ; onderste pixelrij
E0C3   B7              OR    A            ;
E0C4   ED52            SBC   HL,DE        ; controleer Y-coörd.
E0C6   E1              POP   HL           ;
E0C7   C9              RET                ; C = buiten scherm

E0C8   E5      RMDCOL: PUSH  HL           ;
E0C9   21EF00          LD    HL,239       ; max. kolom-waarde
E0CC   B7              OR    A            ;
E0CD   ED42            SBC   HL,BC        ; controleer X-coörd
E0CF   E1              POP   HL           ;
E0D0   C9              RET                ; C = te ver naar rechts

                       END
```

Strings sorteren met "Bubble Sort"

Dit programma sorteert de inhoud van een string-array in stijgende alfabetische volgorde. Het startadres van het array wordt aan de routine doorgegeven als de parameter van de "USR"-functie, bijvoorbeeld : V = USR(VARPTR(A$(Ø))). De grootte van het array speelt geen rol, evenmin als de inhoud ervan, maar het mag slechts één dimensie hebben. Het programma is gebaseerd op het klassieke algoritme van de "bubble sort" : twee strings worden met elkaar vergeleken en indien de tweede kleiner is dan de eerste, worden ze van plaats verwisseld. Een array van 25Ø elementen, die willekeurig werden ingevuld, wordt in ongeveer 2,5 seconden gesorteerd. Een gelijkaardig BASIC programma doet daar meer dan zes minuten over.

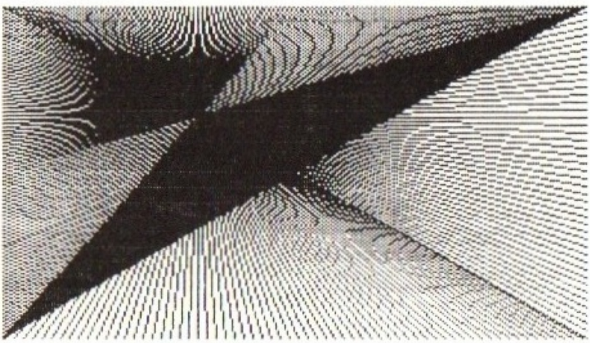
```
                          ORG   OE00H
                          LOAD  OE00H
E000   FE02     SORT:     CP    2         ; geheel getal ?
E002   C0                 RET   NZ        ;
E003   23                 INC   HL        ; HL -> DAC +1
E004   23                 INC   HL        ; HL -> DAC +2
E005   5E                 LD    E,(HL)    ; LSB adres
E006   23                 INC   HL        ; HL -> DAC +3
E007   56                 LD    D,(HL)    ; MSB adres
E008   EB                 EX    DE,HL     ; HL -> A$(0)
E009   E5                 PUSH  HL        ;
E00A   DDE1               POP   IX        ; IX -> A$(0)
E00C   DD7EF8             LD    A,(IX-8)  ; type van array ?
E00F   FE03               CP    3         ; string-array ?
E011   C0                 RET   NZ        ;
E012   DD7EFD             LD    A,(IX-3)  ; dimensie
E015   3D                 DEC   A         ; 1 dimensie ?
E016   C0                 RET   NZ        ;
E017   DD4EFE             LD    C,(IX-2)  ;
E01A   DD46FF             LD    B,(IX-1)  ; BC = aantal elementen
E01D   C5       SR2:      PUSH  BC        ;
E01E   E5                 PUSH  HL        ; HL -> sig.(N)
E01F   46       SR3:      LD    B,(HL)    ; B = lengte (N)
E020   23                 INC   HL
E021   5E                 LD    E,(HL)    ;
E022   23                 INC   HL        ;
E023   E5                 PUSH  HL        ;
E024   56                 LD    D,(HL)    ; DE -> string (N)
E025   23                 INC   HL        ; HL -> sig.(N+1)
E026   4E                 LD    C,(HL)    ; C = lengte (N+1)
E027   23                 INC   HL        ;
E028   7E                 LD    A,(HL)    ;
E029   23                 INC   HL        ;
E02A   E5                 PUSH  HL        ;
E02B   66                 LD    H,(HL)    ;
E02C   6F                 LD    L,A       ; HL -> string (N+1)
E02D   EB                 EX    DE,HL     ; HL -> (N), DE ->(N+1)
E02E   04                 INC   B         ;
E02F   0C                 INC   C         ;
E030   05       SR4:      DEC   B         ; rest  van lengte (N)
E031   2825               JR    Z,NEXT    ; Z=(N)<=(N+1)
E033   0D                 DEC   C         ; rest lengte (N+1)
E034   2808               JR    Z,SWAP    ; Z=(N+1)<(N)
E036   1A                 LD    A,(DE)    ; karakter uit (N+1)
E037   BE                 CP    (HL)      ; karakter uit (N)
E038   13                 INC   DE        ;
E039   23                 INC   HL        ;
E03A   28F4               JR    Z,SR4     ; gelijk, ga door
E03C   301A               JR    NC,NEXT   ; NC = (N)<(N+1)
E03E   E1       SWAP:     POP   HL        ; HL -> sig.(N+1)
E03F   D1                 POP   DE        ; DE -> sig.(N)
E040   0603               LD    B,3       ; lengte signalement
E042   1A       SW2:      LD    A,(DE)    ; verwissel signalementen
E043   4E                 LD    C,(HL)    ;
E044   77                 LD    (HL),A    ;
E045   79                 LD    A,C       ;
E046   12                 LD    (DE),A    ;
E047   1B                 DEC   DE        ;
E048   2B                 DEC   HL        ;
```

```
ADRES  HEXcode LABEL   MNEMONICS       COMMENTAAR

E049   10F7            DJNZ SW2        ;
E04B   DDE5            PUSH IX         ;
E04D   E1              POP  HL         ; HL -> A$(0)
E04E   B7              OR   A          ;
E04F   ED52            SBC  HL,DE      ; start vh array ?
E051   3007            JR   NC,NX2     ; NC = bij start
E053   1B              DEC  DE         ; 1 plaats terug
E054   1B              DEC  DE         ;
E055   EB              EX   DE,HL      ; HL -> sig. (N-1)
E056   18C7            JR   SR3        ; controleer nog eens
E058   E1      NEXT:   POP  HL         ; niet meer nodig
E059   E1              POP  HL         ;
E05A   E1      NX2:    POP  HL         ; HL -> sig. (N)
E05B   C1              POP  BC         ; BC = aantal elementen
E05C   23              INC  HL         ; volgend signalement
E05D   23              INC  HL         ;
E05E   23              INC  HL         ;
E05F   0B              DEC  BC         ;
E060   78              LD   A,B        ;
E061   B1              OR   C          ; klaar ?
E062   20B9            JR   NZ,SR2     ;
E064   C9              RET             ;
```

Afdruk van het scherm op papier

Dit programma zorgt ervoor dat de inhoud van het scherm, in welke mode ook, naar de printer wordt gestuurd. Wanneer het in werking wordt gesteld door een "USR"-statement, zorgt het programma er enkel voor dat het van dan af een deel uitmaakt van de aftasting van het toetsenbord door de interrupt-verwerker, via een "hook". Daarna wordt het programma in feite een deel van de interrupt-routine : door om het even wanneer op de "ESC"-toets te drukken, wordt het programma gestart. Zonodig kan het afdrukken worden onderbroken door de "CTRL/STOP"-toetsen in te drukken. Hieronder staat een voorbeeld van een afdruk :



De eenvoudigste manier om een afdruk van het scherm te krijgen is : alle karaktercodes uit de Namentabel naar de printer copiëren. Dit zou evenwel enkel in de twee tekstmodes werken, de sprites kunnen op die manier niet getoond worden, en het resultaat zou een weergave zijn van de karakterset van de printer in plaats van die van de Video Display Processor. Om dat te omzeilen, wordt het scherm naar de printer gecopieerd als een matrix van 240/256 x 192 bits, waardoor alle modes afgedrukt kunnen worden. Elk puntje in de matrix wordt afgeleid van de kleurcode van het overeenkomstige punt op het scherm. De kleuren 0 tot 7 geven geen punt op de printer, de kleuren 8 tot 15 wel.

De kleurcode van een bepaald punt wordt berekend door eerst de tweeëndertig sprites één na één te bekijken, om te zien of het punt door één ervan wordt overlapt. Indien elke sprite op dat bepaald punt transparant is, wordt het karakter-vlak onderzocht. Met behulp van de coördinaten van het punt wordt in de Namentabel gezocht naar de overeenkomstige positie en dan wordt, via de karaktercode, het relevante bit in het overeenkomstige

patroon bepaald. Blijkt de kleur van dat bit transparant te zijn, dan wordt de achtergrondkleur genomen.
Merk op dat de controlecodes in het programma, bedoeld zijn voor de Epson FX80-printer. Die zijn in de tekst aangeduid, zodat ze voor een andere printer aangepast kunnen worden. Eén groep ervan dient om de "bit image"-mode in te stellen bij de start van een regel van 240/256 bytes (elk byte bepaalt acht puntjes op een vertikale lijn). Een andere groep laat de printer een regel verder bewegen op het einde van een regel. Het programma is geschreven met het oog op maximale snelheid in plaats van zo kort mogelijke code. Het duurt ongeveer vijf seconden plus de tijd die de printer nodig heeft, om de 46.080 of 49.152 puntjes van het beeld te produceren.

SCREEN DUMP PROGRAMMA VOOR ALLE SCHERMEN (0 t/m 3)

```
                                ORG  0E000H
                                LOAD 0E000H

                            ;****************************
                            ;* BIOS STANDAARD ROUTINES  *
                            ;****************************

                            RDVRM:   EQU 004AH
                            CALATR:  EQU 0087H
                            LPTOUT:  EQU 00A5H

                            ;****************************
                            ;*    SYSTEEMVARIABELEN     *
                            ;****************************

                            T32COL:  EQU 0F3BFH
                            GRPNAM:  EQU 0F3C7H
                            GRPCOL:  EQU 0F3C9H
                            GRPCGP:  EQU 0F3CBH
                            MLTNAM:  EQU 0F3D1H
                            MLTCGP:  EQU 0F3D5H
                            RG1SAV:  EQU 0F3E0H
                            RG7SAV:  EQU 0F3E6H
                            NAMBAS:  EQU 0F922H
                            CGPBAS:  EQU 0F924H
                            PATBAS:  EQU 0F926H
                            ATRBAS:  EQU 0F928H
                            SCRMOD:  EQU 0FCAFH
                            HKEYC :  EQU 0FDCCH

                            ;*****************************
                            ;*    CONTROLE KARAKTERS     *
                            ;*****************************

                            CR:      EQU 13
                            ESC:     EQU 27

ADRES HEXcodes  LABEL     MNEMONICS           COMMENTAAR

E000  3ACCFD    ENTRY:    LD   A,(HKEYC)      ; Hook
E003  FEC9                CP   0C9H           ; Vrij voor gebruik?
E005  C0                  RET  NZ             ;
E006  2112E0              LD   HL,DUMP        ; Waar naar toe?
E009  22CDFD              LD   (HKEYC+1),HL   ; Verplaats Hook
E00C  3ECD                LD   A,0CDH         ; CALL
E00E  32CCFD              LD   (HKEYC),A      ;
E011  C9                  RET                 ;

E012  FE3A      DUMP:     CP   3AH            ; ESC toets nummer ?
E014  C0                  RET  NZ             ;
E015  F5                  PUSH AF             ;
E016  C5                  PUSH BC             ;
E017  D5                  PUSH DE             ;
E018  E5                  PUSH HL             ;
E019  ED734FE2            LD   (BRKSTK),SP    ; CTRL/STOP ?
E01D  0E00                LD   C,0            ; C=rij
E01F  3AAFFC    DU1:      LD   A,(SCRMOD)     ; Mode
E022  B7                  OR   A              ;
```

```
ADRES HEXcodes  LABEL   MNEMONICS           COMMENTAAR

E023  21F000            LD   HL,240         ; 40 pixels per rij
E026  112806            LD   DE,6*256+40    ;
E029  2806              JR   Z,DU2          ;
E02B  210001            LD   HL,256         ; T32,GRP,MLT pixels
E02E  112008            LD   DE,8*256+32    ;
E031  3E1B      DU2:    LD   A,ESC          ; ****** FX 80 *****
E033  CD8DE0            CALL PRINT          ; *                *
E036  3E4B              LD   A,"K"          ; *    Bit mode    *
E038  CD8DE0            CALL PRINT          ; *                *
E03B  7D                LD   A,L            ; *   Bytes  LSB   *
E03C  CD8DE0            CALL PRINT          ; *                *
E03F  7C                LD   A,H            ; *   Bytes  MSB   *
E040  CD8DE0            CALL PRINT          ; ******************
E043  0600              LD   B,0            ; B=Kolom
E045  CD97E0    DU3:    CALL CELL           ; Maak een 8x8 cel
E048  D5                PUSH DE             ;
E049  C5                PUSH BC             ;
E04A  2151E2            LD   HL,CBUFF       ; HL->Kleuren
E04D  42                LD   B,D            ; B=pixel kol.(6 of 8
E04E  110800            LD   DE,8           ; CBUFF + 8
E051  C5        DU4:    PUSH BC             ;
E052  E5                PUSH HL             ;
E053  0608              LD   B,8            ; B=Pixel rijen
E055  7E        DU5:    LD   A,(HL)         ; A=Kleur code
E056  FE08              CP   8              ; Donker of Licht ?
E058  3F                CCF                 ; Licht=Print pixel
E059  CB11              RL   C              ;
E05B  19                ADD  HL,DE          ; Volgende pixel rij
E05C  10F7              DJNZ DU5            ;
E05E  79                LD   A,C            ; 8 verticale pixels
E05F  CD8DE0            CALL PRINT          ;
E062  E1                POP  HL             ;
E063  C1                POP  BC             ;
E064  23                INC  HL             ; Volgende pixelkolom
E065  10EA              DJNZ DU4            ;
E067  C1                POP  BC             ;
E068  D1                POP  DE             ;
E069  04                INC  B              ; Volgende kolom
E06A  78                LD   A,B            ;
E06B  BB                CP   E              ; Eind van de rij
E06C  20D7              JR   NZ,DU3         ;
E06E  3E0D              LD   A,CR           ; Printerkop links
E070  CD8DE0            CALL PRINT          ;
E073  3E1B              LD   A,ESC          ; ***** FX 80 *****
E075  CD8DE0            CALL PRINT          ; *               *
E078  3E4A              LD   A,"J"          ; *   Paper Feed  *
E07A  CD8DE0            CALL PRINT          ; *               *
E07D  3E18              LD   A,24           ; * 24/216= 1/9"  *
E07F  CD8DE0            CALL PRINT          ; *****************
E082  0C                INC  C              ; Volgende rij
E083  79                LD   A,C            ;
E084  FE18              CP   24             ; Einde scherm ?
E086  2097              JR   NZ,DU1         ;
E088  E1        DU6:    POP  HL             ;
E089  D1                POP  DE             ;
E08A  C1                POP  BC             ;
E08B  F1                POP  AF             ;
E08C  C9                RET                 ;
```

```
ADRES HEXcodes  LABEL    MNEMONICS          COMMENTAAR

E08D  CDA500    PRINT    CALL LPTOUT        ; Naar printer
E090  D0                 RET  NC            ; CTRL-STOP ?
E091  ED7B4FE2           LD   SP,(BRKSTK)   ; Restore Stack
E095  18F1               JR   DU6           ; Stop programma

E097  C5        CELL:    PUSH BC            ;
E098  D5                 PUSH DE            ;
E099  E5                 PUSH HL            ;
E09A  FDE5               PUSH IY            ;
E09C  2151E2             LD   HL,CBUFF      ;
E09F  3E40               LD   A,64          ;
E0A1  3600      CL1:     LD   (HL),0        ; Transparant
E0A3  23                 INC  HL            ;
E0A4  3D                 DEC  A             ; Fill
E0A5  20FA               JR   NZ,CL1        ;
E0A7  3AAFFC             LD   A,(SCRMOD)    ; Mode ?
E0AA  B7                 OR   A             ; T40 ?
E0AB  F5                 PUSH AF            ;
E0AC  C5                 PUSH BC            ;
E0AD  C469E1             CALL NZ,SPRTES     ; Sprites eerst
E0B0  C1                 POP  BC            ;
E0B1  69                 LD   L,C           ;
E0B2  2600               LD   H,0           ; HL=rij
E0B4  29                 ADD  HL,HL         ;
E0B5  29                 ADD  HL,HL         ;
E0B6  29                 ADD  HL,HL         ; HL=rij*8
E0B7  5D                 LD   E,L           ;
E0B8  54                 LD   D,H           ; DE=rij*8
E0B9  29                 ADD  HL,HL         ;
E0BA  29                 ADD  HL,HL         ; HL=rij*32
E0BB  F1                 POP  AF            ; Mode
E0BC  F5                 PUSH AF            ;
E0BD  2001               JR   NZ,CL2        ; T40?
E0BF  19                 ADD  HL,DE         ; HL=rij*40
E0C0  58        CL2:     LD   E,B           ; DE=Kolom
E0C1  19                 ADD  HL,DE         ;
E0C2  EB                 EX   DE,HL         ; DE=NAMTAB
E0C3  D602               SUB  2             ; Mode
E0C5  79                 LD   A,C           ; A=rij
E0C6  010000             LD   BC,0          ; BC=CGPTAB
E0C9  2A24F9             LD   HL,(CGPBAS)   ;
E0CC  E5                 PUSH HL            ;
E0CD  2A22F9             LD   HL,(NAMBAS)   ;
E0D0  3819               JR   C,CL4         ; C=T40 of T32
E0D2  200C               JR   NZ,CL3        ; NZ=MLT
E0D4  2ACBF3             LD   HL,(GRPCGP)   ; Anders GRP
E0D7  E3                 EX   (SP),HL       ;
E0D8  2AC7F3             LD   HL,(GRPNAM)   ;
E0DB  E618               AND  18H           ; Rij MSBs
E0DD  47                 LD   B,A           ; 1/3=2kB CGP
E0DE  180B               JR   CL4           ;
E0E0  2AD5F3    CL3:     LD   HL,(MLTCGP)   ;
E0E3  E3                 EX   (SP),HL       ;
E0E4  2AD1F3             LD   HL,(MLTNAM)   ;
E0E7  07                 RLCA               ; Rij*2
E0E8  E606               AND  6             ;
E0EA  4F                 LD   C,A           ; 1/6=2kB CGP
E0EB  19        CL4:     ADD  HL,DE         ; HL->NAMTAB
```

```
ADRES HEXcodes   LABEL    MNEMONICS          COMMENTAAR

E0EC  CD4A00              CALL RDVRM         ; HAAL KARAKTER CODE
E0EF  6F                  LD   L,A           ;
E0F0  2600                LD   H,0           ; HL=Karakter code
E0F2  29                  ADD  HL,HL         ;
E0F3  29                  ADD  HL,HL         ;
E0F4  29                  ADD  HL,HL         ; HL=karakt.*8
E0F5  09                  ADD  HL,BC         ; GRP,MLT
E0F6  EB                  EX   DE,HL         ; DE=CGPTAB
E0F7  FDE1                POP  IY            ; IY=CGPTAB startadres
E0F9  FD19                ADD  IY,DE         ; IY->Patroon
E0FB  2AC9F3              LD   HL,(GRPCOL)   ;
E0FE  19                  ADD  HL,DE         ; HL->GRP Kleuren
E0FF  0F                  RRCA               ;
E100  0F                  RRCA               ;
E101  0F                  RRCA               ; Karaktercode/8
E102  E61F                AND  1FH           ;
E104  4F                  LD   C,A           ;
E105  0600                LD   B,0           ;
E107  3AE6F3              LD   A,(RG7SAV)    ; T40 Kleuren
E10A  57                  LD   D,A           ; D=T40 Kleuren
E10B  E60F                AND  0FH           ;
E10D  5F                  LD   E,A           ; E=Achtergrondkleur
E10E  F1                  POP  AF            ; Mode
E10F  E5                  PUSH HL            ; STK->GRP Kleuren
E110  3D                  DEC  A             ;
E111  2008                JR   NZ,CL5        ; Z=T32
E113  2ABFF3              LD   HL,(T32COL)   ;
E116  09                  ADD  HL,BC         ; HL-> T32 Kleuren
E117  CD4A00              CALL RDVRM         ; Haal T32 Kleuren
E11A  57                  LD   D,A           ; D=T32 Kleuren
E11B  2151E2     CL5:     LD   HL,(CBUFF)    ; Resultaten
E11E  0608                LD   B,8           ; Pixel Rijen
E120  FDE5       CL6:     PUSH IY            ;
E122  E3                  EX   (SP),HL       ; HL->Patroon
E123  CD4A00              CALL RDVRM         ; Haal Patroon
E126  4F                  LD   C,A           ; C=Patroon
E127  E1                  POP  HL            ;
E128  FD23                INC  IY            ; Volgende pixelrij
E12A  3AAFFC              LD   A,(SCRMOD)    ; Mode
E12D  D602                SUB  2             ;
E12F  3815                JR   C,CL8         ; C=T40 of T32
E131  280C                JR   Z,CL7         ; Z=GRP
E133  51                  LD   D,C           ; MLT Kleuren=patroon
E134  0EF0                LD   C,0F0H        ; Dummy MLT patroon
E136  78                  LD   A,B           ; Pixelrij
E137  FE05                CP   5             ; Helft van CELL ?
E139  280B                JR   Z,CL8         ;
E13B  FD2B                DEC  IY            ;
E13D  1807                JR   CL8           ;
E13F  E3         CL7:     EX   (SP),HL       ; HL->GRP Kleuren
E140  CD4A00              CALL RDVRM         ; Haal Kleuren
E143  57                  LD   D,A           ; D=GRP Kleuren
E144  23                  INC  HL            ; Volgende pixelrij
E145  E3                  EX   (SP),HL       ; STK->GRP Kleuren
E146  C5         CL8:     PUSH BC            ;
E147  0608                LD   B,8           ; Pixel kolommen
E149  CB11       CL9:     RL   C             ; Pixel patroon
E14B  34                  INC  (HL)          ;
```

```
ADRES HEXcodes   LABEL     MNEMONICS        COMMENTAAR

E14C  35                   DEC  (HL)        ; Controleer CBUFF
E14D  200D                 JR   NZ,CL12     ; NZ=SPRITE aanwezig
E14F  7A                   LD   A,D         ; A=Kleuren
E150  3004                 JR   NC,CL10     ; NC=0 Pixel
E152  0F                   RRCA             ;
E153  0F                   RRCA             ;
E154  0F                   RRCA             ;
E155  0F                   RRCA             ; Kies 1 Kleur
E156  E60F       CL10:     AND  0FH         ;
E158  2001                 JR   NZ,CL11     ; Z=Transparant
E15A  7B                   LD   A,E         ; Gebruik achtergrond
E15B  77         CL11:     LD   (HL),A      ; Kleur in CBUFF
E15C  23         CL12:     INC  HL          ;
E15D  10EA                 DJNZ CL9         ; Volgende pixel kolom
E15F  C1                   POP  BC          ;
E160  10BE                 DJNZ CL6         ; Volgende pixel rij
E162  E1                   POP  HL          ;
E163  FDE1                 POP  IY          ;
E165  E1                   POP  HL          ;
E166  D1                   POP  DE          ;
E167  C1                   POP  BC          ;
E168  C9                   RET              ;
                                            ; A=Kolom
E169  78         SPRTES    LD   A,B         ;
E16A  07                   RLCA             ;
E16B  07                   RLCA             ; A=X coördinaat
E16C  07                   RLCA             ; RH rand van karaktercel
E16D  C607                 ADD  A,7         ; B=X coördinaat
E16F  47                   LD   B,A         ; A=rij
E170  79                   LD   A,C         ;
E171  07                   RLCA             ;
E172  07                   RLCA             ; A=Y coördinaat
E173  07                   RLCA             ; Onderkant karaktercel
E174  C607                 ADD  A,7         ; C=Y coördinaat
E176  4F                   LD   C,A         ; Sprite nummer
E177  AF                   XOR  A           ; HL->Attributen
E178  CD8700     SS1:      CALL CALATR      ; D=Sprite nummer
E17B  57                   LD   D,A         ; Haal Sprite Y
E17C  CD4A00               CALL RDVRM       ; Einde-markering?
E17F  FED0                 CP   208         ;
E181  C8                   RET  Z           ;
E182  D5                   PUSH DE          ;
E183  C5                   PUSH BC          ; Verwerk sprite
E184  CD8FE1               CALL SPRITE      ;
E187  C1                   POP  BC          ;
E188  F1                   POP  AF          ; Volgende sprite nummer
E189  3C                   INC  A           ; Alles gehad?
E18A  FE20                 CP   32          ;
E18C  20EA                 JR   NZ,SS1      ;
E18E  C9                   RET

E18F  91         SPRITE:   SUB  C           ; (SY-Y)
E190  2F                   CPL              ; Maak (Y-SY)
E191  FE27                 CP   39          ; Overlapping ?
E193  D0                   RET  NC          ;
E194  4F                   LD   C,A         ; C=(Y-SY)
E195  23                   INC  HL          ;
E196  CD4A00               CALL RDVRM       ; Haal Sprite X
```

```
ADRES HEXcodes   LABEL    MNEMONICS        COMMENTAAR

E199  5F                  LD   E,A         ;
E19A  78                  LD   A,B         ; A=X coördinaat
E19B  93                  SUB  E           ;
E19C  5F                  LD   E,A         ; E=(X-SX)
E19D  9F                  SBC  A,A         ; Maak het 16 bits
E19E  57                  LD   D,A         ; DE=(X-SX)
E19F  23                  INC  HL          ;
E1A0  CD4A00              CALL RDVRM       ; Haal patroon-nummer
E1A3  47                  LD   B,A         ;
E1A4  23                  INC  HL          ;
E1A5  CD4A00              CALL RDVRM       ; Haal EC en Kleur
E1A8  CB7F                BIT  7,A         ; "early clock?"
E1AA  2805                JR   Z,SP1       ;
E1AC  212000              LD   HL,32       ;
E1AF  19                  ADD  HL,DE       ; Verhoog (X-SX)
E1B0  EB                  EX   DE,HL       ;
E1B1  14         SP1:     INC  D           ;
E1B2  15                  DEC  D           ; (X-SX)>255 of negatief
E1B3  C0                  RET  NZ          ; NZ=Buiten cell
E1B4  E60F                AND  0FH         ; Kleur
E1B6  C8                  RET  Z           ; Z=Transparant
E1B7  57                  LD   D,A         ; D=Kleur
E1B8  3AE0F3              LD   A,(RG1SAV)  ; Vlaggen
E1BB  CB4F                BIT  1,A         ; Afmeting
E1BD  0F                  RRCA             ; Vergrotingsfactor
E1BE  3E08                LD   A,8         ; Minimale afmeting
E1C0  3001                JR   NC,SP2      ;
E1C2  87                  ADD  A,A         ; Dubbel voor Vergroting
E1C3  2805       SP2:     JR   Z,SP3       ;
E1C5  CB80                RES  0,B         ; Verander patroon-nummer
E1C7  CB88                RES  1,B         ;
E1C9  87                  ADD  A,A         ; Dubbel voor SIZE
E1CA  6F         SP3:     LD   L,A         ; L=Sprite grootte
E1CB  C606                ADD  A,6         ;
E1CD  B9                  CP   C           ;
E1CE  D8                  RET  C           ; Sprite aanwezig
E1CF  BB                  CP   E           ;
E1D0  D8                  RET  C           ; Sprite naar links
E1D1  79                  LD   A,C         ;
E1D2  D607                SUB  7           ; (Y-SY) vanaf de top
E1D4  4F                  LD   C,A         ;
E1D5  7D                  LD   A,L         ; A=Sprite grootte
E1D6  2608                LD   H,8         ; Max pixel rijen
E1D8  3808                JR   C,SP5       ; C=Onder top karaktercel
E1DA  91                  SUB  C           ; A=Overlapping Pixelrij
E1DB  FE09                CP   9           ;
E1DD  3802                JR   C,SP4       ;
E1DF  3E08                LD   A,8         ;
E1E1  67         SP4:     LD   H,A         ; H=Rij overlapping
E1E2  7B         SP5:     LD   A,E         ;
E1E3  D607                SUB  7           ; (X-SX) van linkerkarakter
E1E5  5F                  LD   E,A         ;
E1E6  7D                  LD   A,L         ; A=Sprite grootte
E1E7  2E08                LD   L,8         ; Max. Pixelkolommen
E1E9  3808                JR   C,SP7       ; C=Linkerkarakter voorbij
E1EB  93                  SUB  E           ; A=Pixelkolom overlapping
E1EC  FE09                CP   9           ;
E1EE  3802                JR   C,SP6       ;
```

```
ADRES HEXcodes   LABEL     MNEMONICS           COMMENTAAR

E1F0  3E08                 LD   A,8            ;
E1F2  6F         SP6:      LD   L,A            ; L-kolom overlapping
E1F3  FD2151E2   SP7:      LD   IY,CBUFF       ; Resultaten
E1F7  D5         SP8:      PUSH DE             ;
E1F8  CB79                 BIT  7,C            ; Sprite bereikt ?
E1FA  2048                 JR   NZ,SP15        ;
E1FC  E5                   PUSH HL             ;
E1FD  FDE5                 PUSH IY             ;
E1FF  CB7B       SP9:      BIT  7,E            ; Sprite bereikt ?
E201  2038                 JR   NZ,SP14        ;
E203  FD7E00               LD   A,(IY+0)       ; CBUFF
E206  B7                   OR   A              ; Transparant ?
E207  2032                 JR   NZ,SP14        ;
E209  C5                   PUSH BC             ;
E20A  D5                   PUSH DE             ;
E20B  E5                   PUSH HL             ;
E20C  3AE0F3               LD   A,(RG1SAV)     ; Vlaggen
E20F  0F                   RRCA                ; Vergrotingsfactor
E210  3004                 JR   NC,SP10        ;
E212  CB39                 SRL  C              ; (Y-SY)/2
E214  CB3B                 SRL  E              ; (X-SX)/2
E216  CB5B       SP10:     BIT  3,E            ; (X-SX)>7?
E218  2804                 JR   Z,SP11         ;
E21A  CB9B                 RES  3,E            ; (X-SX)-8
E21C  CBE1                 SET  4,C            ; (Y-SY)+16
E21E  68         SP11:     LD   L,B            ;
E21F  2600                 LD   H,0            ; HL=patroon-nummer
E221  44                   LD   B,H            ; BC=Y
E222  29                   ADD  HL,HL          ;
E223  29                   ADD  HL,HL          ;
E224  29                   ADD  HL,HL          ; HL=patroon*8
E225  09                   ADD  HL,BC          ; Kies pixel rij
E226  ED4B26F9             LD   BC,(PATBAS)    ;
E22A  09                   ADD  HL,BC          ; HL->Patroon
E22B  CD4A00               CALL RDVRM          ; Haal pixel rij
E22E  1C                   INC  E              ;
E22F  07         SP12:     RLCA                ; Kies pixel kolom
E230  1D                   DEC  E              ;
E231  20FC                 JR   NZ,SP12        ;
E233  3003                 JR   NC,SP13        ; NC=0 pixel
E235  FD7200               LD   (IY+0),D       ; Kleur in CBUFF
E238  E1         SP13:     POP  HL             ;
E239  D1                   POP  DE             ;
E23A  C1                   POP  BC             ;
E23B  FD23       SP14      INC  IY             ;
E23D  1C                   INC  E              ; 1 pixelkolom rechts
E23E  2D                   DEC  L              ; Kolommen klaar?
E23F  20BE                 JR   NZ,SP9         ;
E241  FDE1                 POP  IY             ;
E243  E1                   POP  HL             ;
E244  110800     SP15:     LD   DE,8           ;
E247  FD19                 ADD  IY,DE          ;
E249  D1                   POP  DE             ;
E24A  0C                   INC  C              ;1 pixelrij omlaag
E24B  25                   DEC  H              ;klaar ?
E24C  20A9                 JR   NZ,SP8         ;
E24E  C9                   RET                 ;
E24F  0000       BRKSTK:   DEFW 0              ; BREAK STACK
E251             CBUFF:    DEFS 64             ; CELL BUFFER
```

Andere karakterset maken

Dit programma maakt het mogelijk om de karakterpatronen van de MSX te wijzigen. Bij de start van het programma wordt de karakterset van 2 Kb gecopieerd vanuit zijn huidige positie (gewoonlijk de ROM) naar de buffer CHRTAB (E2A3H tot EAA2H). Daarna wordt dit op het scherm gezet :



Het programma werkt op twee niveaus : commando en edit. De "RETURN"-toets schakelt om tussen deze twee niveaus. In de commando-mode worden de vier cursortoetsen gebruikt om een karakter te kiezen dat gewijzigd dient te worden. Dit wordt door een grote cursor aangegeven en wordt tevens, vergroot, op de rechterkant van het scherm afgedrukt. Een druk op de "Q"-toets ("quit") doet het programma stoppen en geeft de controle weer over aan het BASIC systeem. Een druk op de "A"-toets ("adopt") maakt dat de nieuwe karakterset door het systeem wordt beschouwd als zijn eigen karakterset. Dat houdt in dat de karakterset naar het hoogste deel van het geheugen wordt gecopieerd (EB80H tot F37FH), en dat zijn Slot-identificator en adres in CGPNT wordt gezet.

In "edit"-mode dienen de cursortoetsen om de pixel aan te geven die gewijzigd moet worden. Die wordt door een kleine cursor aangegeven. Een druk op de "SPACE"-toets wist de pixel uit, een druk op de "."-toets zet een pixel op die plaats. Samen met het grote karakter rechts, wordt ook het karakter in het rooster links aangepast.

De karakterset in CHRTAB kan op cassette gezet worden met het "BSAVE"-statement en later terug worden ingeladen met het

"BLOAD"-statement. De subroutine "ADOPT" in het programma moet met het karakterblok meege"SAVE"d worden, en bij het inladen opnieuw aangeroepen, zodat het systeem opnieuw die nieuwe karakterset als de zijne herkent. Een andere mogelijkheid is, de karakterset alleen op cassette zetten en bij het opnieuw inladen ervan, de Slot-identificator en het adres via BASIC statements in CGPNT zetten. Het wijzigen van de karakterpatronen heeft absoluut geen invloed op de werking van het MSX-systeem.

Maken van andere karakterset

```
                ORG     OE000H
                LOAD    OE000H

        ;***************************
        ;*  BIOS STANDAARDROUTINES *
        ;***************************

        RDSLT:  EQU     000CH
        RDVRM:  EQU     004AH
        WRTVRM: EQU     004DH
        FILVRM: EQU     0056H
        INIGRP: EQU     0072H
        CHSNS:  EQU     009CH
        CHGET:  EQU     009FH
        MAPXYC: EQU     0111H
        FETCHC: EQU     0114H
        RSLREG: EQU     0138H

        ;***************************
        ;* VARIABELEN IN WERKGEBIED *
        ;***************************

        GRPCOL: EQU     0F3C9H
        FORCLR: EQU     0F3E9H
        BAKCLR: EQU     0F3EAH
        CGPNT:  EQU     0F91FH
        EXPTBL: EQU     0FCC1H
        SLTTBL: EQU     0FCC5H

        ;***************************
        ;* CONTROLE-KARAKTERS      *
        ;***************************

          CR:     EQU     13
          RIGHT:  EQU     28
          LEFT:   EQU     29
          UP:     EQU     30
          DOWN:   EQU     31


ADRES   HEXcode LABEL    MNEMONICS       COMMENTAAR

E000    CDF6E0  CHEDIT:  CALL INIT       ; koude start
E003    CDBDE0  CH1:     CALL CHRMAG     ; vergroot karakter
E006    CDFEE1           CALL CHRXY      ; kar.-coordinaten
E009    1608             LD   D,8        ; grootte cursor
E00B    CD2FE2           CALL GETKEY     ; welk commando ?
E00E    FE51             CP   "Q"        ; "quit", stoppen
E010    C8               RET  Z          ;
E011    2103E0           LD   HL,CH1     ; zet "RET" klaar
E014    E5               PUSH HL         ;
E015    FE41             CP   "A"        ; "adopt"
E017    CA6EE2           JP   Z,ADOPT    ;
```

```
ADRES  HEXcode LABEL   MNEMONICS        COMMENTAAR

E01A   FE0D            CP   CR          ; "edit"
E01C   281F            JR   Z,EDIT      ;
E01E   0E01            LD   C,1         ; C = offset
E020   FE1C            CP   RIGHT       ; rechts
E022   2811            JR   Z,CH2       ;
E024   0EFF            LD   C,0FFH      ;
E026   FE1D            CP   LEFT        ; links
E028   280B            JR   Z,CH2       ;
E02A   0EF0            LD   C,0F0H      ;
E02C   FE1E            CP   UP          ; omhoog
E02E   2805            JR   Z,CH2       ;
E030   0E10            LD   C,16        ;
E032   FE1F            CP   DOWN        ; omlaag
E034   C0              RET  NZ          ;
E035   3AA1E2  CH2:    LD   A,(CHRNUM)  ; huidig karakter
E038   81              ADD  A,C         ; + offset
E039   32A1E2          LD   (CHRNUM),A  ; = nieuw karakter
E03C   C9              RET              ;

E03D   CDE6E1  EDIT:   CALL DOTXY       ; pixel-coordinaten
E040   1602            LD   A,2         ; cursorgrootte
E042   CD2FE2          CALL GETKEY      ; welk commando ?
E045   FE0D            CP   CR          ; stoppen ?
E047   C8              RET  Z           ;
E048   213DE0          LD   HL,EDIT     ; "RET"-adres
E04B   E5              PUSH HL          ;
E04C   0100FE          LD   BC,0FE00H   ; AND/OR-maskers
E04F   FE20            CP   " "         ; spatie
E051   2824            JR   Z,ED3       ;
E053   0C              INC  C           ; nieuw OR-masker
E054   FE2E            CP   "."         ; punt
E056   281F            JR   Z,ED3       ;
E058   FE1C            CP   RIGHT       ; rechts
E05A   2811            JR   Z,ED2       ;
E05C   0EFF            LD   C,0FFH      ; C = offset
E05E   FE1D            CP   LEFT        ; links
E060   280B            JR   Z,ED2       ;
E062   0EF8            LD   C,0F8H      ;
E064   FE1E            CP   UP          ; omhoog
E066   2805            JR   Z,ED2       ;
E068   0E08            LD   C,8         ;
E06A   FE1F            CP   DOWN        ; omlaag
E06C   C0              RET  NZ          ;
E06D   3AA2E2  ED2:    LD   A,(DOTNUM)  ; huidige pixel
E070   81              ADD  A,C         ; + offset
E071   E63F            AND  63          ; doortellen
E073   32A2E2          LD   (DOTNUM),A  ; nieuwe pixel
E076   C9              RET              ;
E077   CD1EE2  ED3:    CALL PATPOS      ; IY-> patroon
E07A   3AA2E2          LD   A,(DOTNUM)  ; huidige pixel
E07D   F5              PUSH AF          ;
E07E   0F              RRCA             ;
E07F   0F              RRCA             ;
E080   0F              RRCA             ;
E081   E607            AND  7           ; A = pixel-lijn
E083   5F              LD   E,A         ;
E084   1600            LD   D,0         ; DE = pixel-lijn
E086   FD19            ADD  IY,DE       ; IY-> lijn
```

```
ADRES  HEXcode LABEL   MNEMONICS          COMMENTAAR

E088   F1              POP  AF            ;
E089   E607            AND  7             ; A = kolom
E08B   3C              INC  A             ;
E08C   CB08    ED4:    RRC  B             ; AND-masker
E08E   CB09            RRC  C             ; OR-masker
E090   3D              DEC  A             ; tel kolommen
E091   20F9            JR   NZ,ED4        ;
E093   FD7E00          LD   A,(IY+10)     ; A = patroon
E096   A0              AND  B             ; oud bit eruit halen
E097   B1              OR   C             ; nieuw bit erin
E098   FD7700          LD   (IY+10),A     ; nieuw patroon
E09B   CDBDE0          CALL CHRMAG        ; pas vergroting aan
E09E   CD1EE2  CHROUT: CALL PATPOS        ; IY-> patroon
E0A1   CDFEE1          CALL CHRXY         ; haal coordinaten
E0A4   CDA3E1          CALL MAP           ; schermadres ?
E0A7   0608            LD   B,8           ; pixel-lijnen
E0A9   D5      CO1:    PUSH DE            ;
E0AA   E5              PUSH HL            ;
E0AB   3E08            LD   A,8           ; pixel-kolommen
E0AD   FD5E00          LD   E,(IY+10)     ; E = patroon
E0B0   CDC4E1          CALL SETROW        ; vul in
E0B3   E1              POP  HL            ; HL = CLOC
E0B4   D1              POP  DE            ; D = CMASK
E0B5   CDB8E1          CALL DOWNP         ; 1 pixel omlaag
E0B8   FD23            INC  IY            ;
E0BA   10ED            DJNZ CO1           ;
E0BC   C9              RET                ;

E0BD   CD1EE2  CHRMAG: CALL PATPOS        ; IY-> patroon
E0C0   0EBF            LD   C,191         ; start van X
E0C2   1E07            LD   E,7           ; start van Y
E0C4   CDA3E1          CALL MAP           ; schermadres ?
E0C7   0608            LD   B,8           ; pixel-lijnen
E0C9   0E05    CM1:    LD   C,5           ; vergroting van lijn
E0CB   C5      CM2:    PUSH BC            ;
E0CC   D5              PUSH DE            ;
E0CD   E5              PUSH HL            ;
E0CE   0608            LD   B,8           ; pixel-kolommen
E0D0   FD7E00          LD   A,(IY+10)     ; A = patroon
E0D3   07      CM3:    RLCA               ; test bit
E0D4   F5              PUSH AF            ;
E0D5   9F              SBC  A,A           ; 0 = 00H, 1 = FFH
E0D6   5F              LD   E,A           ; E= patroon v vergroting
E0D7   3E05            LD   A,5           ; vergroting van kolom
E0D9   CDC4E1          CALL SETROW        ; vul rij in
E0DC   CDAEE1          CALL RIGHTP        ; 1 pixel naar rechts
E0DF   CDAEE1          CALL RIGHTP        ; sla raster over
E0E2   F1              POP  AF            ;
E0E3   10EE            DJNZ CM3           ;
E0E5   E1              POP  HL            ; HL = CLOC
E0E6   D1              POP  DE            ; D = CMASK
E0E7   C1              POP  BC            ;
E0E8   CDB8E1          CALL DOWNP         ; 1 pixel omlaag
E0EB   0D              DEC  C             ;
E0EC   20DD            JR   NZ,CM2        ;
E0EE   CDB8E1          CALL DOWNP         ; sla raster over
E0F1   FD23            INC  IY            ;
E0F3   10D4            DJNZ CM1           ;
E0F5   C9              RET                ;
```

```
ADRES  HEXcode LABEL   MNEMONICS           COMMENTAAR

E0F6   010008  INIT:   LD   BC,2048      ; grootte
E0F9   11A3E2          LD   DE,CHRTAB    ; doeladres
E0FC   2A20F9          LD   HL,(CGPNT+1); bron-adres
E0FF   C5      IN1:    PUSH BC           ;
E100   D5              PUSH DE           ;
E101   3A1FF9          LD   A,(CGPNT)    ; slot-identificator
E104   CD0C00          CALL RDSLT        ; lees karakterpatroon
E107   FB              EI                ;
E108   D1              POP  DE           ;
E109   C1              POP  BC           ;
E10A   12              LD   (DE),A       ; zet in buffer
E10B   13              INC  DE           ;
E10C   23              INC  HL           ;
E10D   0B              DEC  BC           ;
E10E   78              LD   A,B          ;
E10F   B1              OR   C            ;
E110   20ED            JR   NZ,IN1       ;
E112   CD7200          CALL INIGRP       ; SCREEN 2
E115   3AE9F3          LD   A,(FORCLR)   ; kleur 1
E118   07              RLCA              ;
E119   07              RLCA              ;
E11A   07              RLCA              ;
E11B   07              RLCA              ;
E11C   4F              LD   C,A          ; C = kleur 1
E11D   3AEAF3          LD   A,(BAKCLR)   ; kleur 0
E120   B1              OR   C            ; meng
E121   010018          LD   BC,6144      ; lengte kleurentabel
E124   2AC9F3          LD   HL,(GRPCOL); kleurentabel
E127   CD5600          CALL FILVRM       ; vul kleuren in
E12A   210BB1          LD   HL,177*256+11;
E12D   010AFF          LD   BC,0FFH*256+10;
E130   1E06            LD   E,6          ;
E132   3E11            LD   A,17         ;
E134   CD62E1          CALL GRID         ; teken raster
E137   210631          LD   HL,49*256+6;
E13A   01BEAA          LD   BC,0AAH*256+190;
E13D   1E06            LD   E,6          ;
E13F   3E09            LD   A,9          ;
E141   CD62E1          CALL GRID         ; raster rond vergroting
E144   213031          LD   HL,49*256+48;
E147   01BEFF          LD   BC,0FFH*256+190;
E14A   1E06            LD   E,6          ;
E14C   3E02            LD   A,2          ;
E14E   CD62E1          CALL GRID         ; omtrek vd vergroting
E151   AF              XOR  A            ;
E152   32A2E2          LD   (DOTNUM),A   ; huidige pixel
E155   21A1E2          LD   HL,CHRNUM    ;
E158   77              LD   (HL),A       ; huidige karakter
E159   E5      IN2:    PUSH HL           ;
E15A   CD9EE0          CALL CHROUT       ; teken karakter
E15D   E1              POP  HL           ;
E15E   34              INC  (HL)         ; volgende karakter
E15F   20F8            JR   NZ,IN2       ; alle 256
E161   C9              RET               ;
```

```
ADRES  HEXcode LABEL   MNEMONICS        COMMENTAAR

E162   F5      GRID:   PUSH AF          ;
E163   C5              PUSH BC          ;
E164   E5              PUSH HL          ;
E165   CDA3E1          CALL MAP         ; schermadres ?
E168   C1              POP  BC          ; B = lengte, C = step
E169   F1              POP  AF          ;
E16A   5F              LD   E,A         ; E = patroon
E16B   F1              PUSH AF          ;
E16C   F5              PUSH DE          ;
E16E   D5              PUSH HL          ;
E16F   F5      GR1:    PUSH AF          ;
E170   C5              PUSH BC          ;
E171   D5              PUSH DE          ;
E172   E5              PUSH HL          ;
E173   78              LD   A,B         ; A = lengte
E174   CDC4E1          CALL SETROW      ; horizontale lijn
E177   E1              POP  HL          ; HL = CLOC
E178   D1              POP  DE          ; D = CMASK
E179   CDB8E1  GR3:    CALL DOWNP       ; 1 pixel omlaag
E17C   0D              DEC  C           ; step klaar ?
E17D   20FA            JR   NZ,GR3      ;
E17F   C1              POP  BC          ;
E180   F1              POP  AF          ; A = aantal
E181   3D              DEC  A           ; lijnen afgewerkt ?
E182   20EB            JR   NZ,GR1      ;
E184   E1              POP  HL          ; HL = oorspronk. CLOC
E185   D1              POP  DE          ; D = oorspronk. CMASK
E186   F1              POP  AF          ; A = aantal
E17    F5      GR4:    PUSH AF          ;
E188   C5              PUSH BC          ;
E189   D5              PUSH DE          ;
E18A   E5              PUSH HL          ;
E18B   3E01    GR5:    LD   A,1         ; breedte van de lijn
E18D   CDC4E1          CALL SETROW      ; dunne lijn
E190   CDB8E1          CALL DOWNP       ; 1 pixel omlaag
E193   10F6            DJNZ GR5         ; vertikale lengte
E195   E1              POP  HL          ; HL = CLOC
E196   D1              POP  DE          ; D = CMASK
E197   CDAEE1  GR6:    CALL RIGHTP      ; 1 pixel naar rechts
E19A   0D              DEC  C           ; step klaar ?
E19B   20FA            JR   NZ,GR6      ;
E19D   C1              POP  BC          ;
E19E   F1              POP  AF          ; A = aantal
E19F   3D              DEC  A           ; lijnen afgewerkt ?
E1A0   20E5            JR   NZ,GR4      ;
E1A2   C9              RET              ;

E1A3   0600    MAP:    LD   B,0         ; MSB van X
E1A5   50              LD   D,B         ; MSB van Y
E1A6   CD1101          CALL MAPXYC      ; bereken coordinaten
E1A9   CD1401          CALL FETCHC      ; HL = CLOC
E1AC   57              LD   D,A         ; D = CMASK
E1AD   C9              RET              ;

E1AE   CB0A    RIGHTP: RRC  D           ; schuif masker
E1B0   D0              RET  NC          ; NC = zelfde kar.cel
E1B1   C5      RP1:    PUSH BC          ;
E1B2   010800          LD   BC,8        ; offset
```

```
ADRES  HEXcode LABEL   MNEMONICS          COMMENTAAR

E1B5   09              ADD  HL,BC       ; HL = volgende kar.cel
E1B6   C1              POP  BC          ;
E1B7   C9              RET              ;

E1B8   23      DOWNP:  INC  HL          ; CLOC omlaag
E1B9   7D              LD   A,L         ;
E1BA   E607            AND  7           ; kies pixel-lijn
E1BC   C0              RET  NZ          ; NZ = zelfde kar. cel
E1BD   C5              PUSH BC          ;
E1BE   01F800          LD   BC,00F8H    ; offset
E1C1   09              ADD  HL,BC       ; HL = volgende kar.cel
E1C2   C1              POP  BC          ;
E1C3   C9              RET              ;

E1C4   C5      SETROW: PUSH BC          ;
E1C5   47              LD   B,A         ; B = aantal
E1C6   CD4A00  SE1:    CALL RDVRM       ; haal oude patroon
E1C9   4F      SE2:    LD   C,A         ; C = oud
E1CA   7A              LD   A,D         ; A = CMASK
E1CB   2F              CPL              ; AND-masker
E1CC   A1              AND  C           ; oud bit eruit
E1CD   CB03            RLC  E           ; schuif patroon
E1CF   3001            JR   NC,SE3      ; NC = "0"-pixel
E1D1   B2              OR   D           ; set "1"-pixel
E1D2   05      SE3:    DEC  B           ; klaar ?
E1D3   280C            JR   Z,SE4       ;
E1D5   CB0A            RRC  D           ; CMASK naar rechts
E1D7   30F000          JR   NC,SE2      ; NC = zelfde kar. cel
E1D9   CD4D00          CALL WRTVRM      ; pas kar. cel aan
E1DC   CDB1E1          CALL RP1         ; volgende kar. cel
E1DF   18E5            JR   SE1         ; opnieuw
E1E1   CD4D00  SE4:    CALL WRTVRM      ; pas kar. cel aan
E1E4   C1              POP  BC          ;
E1E5   C9              RET              ;

E1E6   3AA2E2  DOTXY:  LD   A,(DOTNUM)  ; huidige pixel
E1E9   F5              PUSH AF          ;
E1EA   E607            AND  7           ; kolom
E1EC   07              RLCA             ;
E1ED   4F              LD   C,A         ; C = kolom*2
E1EE   07              RLCA             ; A = kolom*4
E1EF   81              ADD  A,C         ; A = kolom*6
E1F0   C6BF            ADD  A,191       ; start vh raster
E1F2   4F              LD   C,A         ; C = X-coordinaat
E1F3   F1              POP  AF          ;
E1F4   E638            AND  38H         ; rij*8
E1F6   0F              RRCA             ;
E1F7   5F              LD   E,A         ; E = rij*4
E1F8   0F              RRCA             ; A = rij*2
E1F9   83              ADD  A,E         ; A = rij*6
E1FA   C607            ADD  A,7         ; start vh raster
E1FC   5F              LD   E,A         ; E = Y-coordinaat
E1FD   C9              RET              ;

E1FE   3AA1E2  CHRXY:  LD   A,(CHRNUM)  ; huidig karakter
E201   F5              PUSH AF          ;
E202   CD14E2          CALL MULT11      ; kolom*11
E205   C60C            ADD  A,12        ; start vh raster
```

```
ADRES  HEXcode LABEL    MNEMONICS          COMMENTAAR

E207   4F               LD    C,A          ; C - X-coordinaat
E208   F1               POP   AF           ;
E209   0F               RRCA               ;
E20A   0F               RRCA               ;
E20B   0F               RRCA               ;
E20C   0F               RRCA               ;
E20D   CD14E2           CALL  MULT11       ; rij*11
E210   C608             ADD   A,8          ; start vh raster
E212   5F               LD    E,A          ; E - Y-coordinaat
E213   C9               RET                ;

E214   E60F    MULT11:  AND   0FH          ;
E216   57               LD    D,A          ; D - N
E217   07               RLCA               ;
E218   47               LD    B,A          ; B - N*2
E219   07               RLCA               ;
E21A   07               RLCA               ; A - N*8
E21B   80               ADD   A,B          ;
E21C   82               ADD   A,D          ; A - N*11
E21D   C9               RET                ;

E21E   3AA1E2  PATPOS:  LD    A,(CHRNUM)   ; huidig karakter
E221   6F               LD    L,A          ;
E222   2600             LD    H,0          ; HL - karakter
E224   29               ADD   HL,HL        ;
E225   29               ADD   HL,HL        ;
E226   29               ADD   HL,HL        ; HL - kar.*8
E227   EB               EX    DE,HL        ; DE - kar.*8
E228   FD21A3E2         LD    IY,CHRTAB    ; patronen
E22C   FD19             ADD   IY,DE        ; IY-> patroon
E22E   C9               RET                ;

E22F   0600    GETKEY:  LD    B,0          ; vlag voor cursor
E231   C5      GE1:     PUSH  BC           ; C - X-coordinaat
E232   D5               PUSH  DE           ; E - Y-coordinaat
E233   CD50E2           CALL  INVERT       ; wissel cursor om
E236   D1               POP   DE           ;
E237   C1               POP   BC           ;
E238   04               INC   B            ; wissel vlag
E239   21401F           LD    HL,8000      ; knipper-frekwentie
E23C   CD9C00  GE2:     CALL  CHSNS        ; controleer KEYBUF
E23F   2007             JR    NZ,GE3       ; NZ - toets ingedrukt
E241   2B               DEC   HL           ;
E242   7C               LD    A,H          ;
E243   B5               OR    L            ;
E244   20F6             JR    NZ,GE2       ;
E246   18E9             JR    GE1          ; tijd voor cursor
E248   CB40    GE3:     BIT   0,B          ; hoe is cursor ?
E24A   C450E2           CALL  NZ,INVERT    ; haal cursor weg
E24D   C39F00           JP    CHGET        ; haal karakter op

E250   D5      INVERT:  PUSH  DE           ;
E251   CDA3E1           CALL  MAP          ; schermadres ?
E254   F1               POP   AF           ; A - grootte v cursor
E255   47               LD    B,A          ; B - lijnen
E256   5F               LD    E,A          ; E - kolommen
E257   D5      IV1:     PUSH  DE           ;
E258   E5               PUSH  HL           ;
```

```
ADRES  HEXcode LABEL   MNEMONICS           COMMENTAAR

E259   CD4A00  IV2:    CALL RDVRM          ; oud patroon
E25C   AA              XOR  D              ; wissel een bit om
E25D   CD4D00          CALL WRTVRM         ; zet het terug
E260   CDAEE1          CALL RIGHTP         ; 1 pixel naar rechts
E263   1D              DEC  E              ;
E264   20F3            JR   NZ,IV2         ;
E266   E1              POP  HL             ; HL = CLOC
E267   D1              POP  DE             ; D = CMASK
E268   CDB8E1          CALL DOWNP          ; 1 pixel omlaag
E26B   10EA            DJNZ IV1            ;
E26D   C9              RET                 ;

E26E   010008  ADOPT:  LD   BC,2048        ; grootte
E271   1180EB          LD   DE,0EB08H      ; naar waar
E274   ED5320F9        LD   (CGPNT+1),DE;
E278   21A3E2          LD   HL,CHRTAB      ; van waar
E27B   EDB0            LDIR                ; verzet
E27D   CD3801          CALL RSLREG         ; lees PSLOT-register
E280   07              RLCA                ;
E281   07              RLCA                ;
E282   E603            AND  3              ; selecteer blz. 3
E284   4F              LD   C,A            ;
E285   0600            LD   B,0            ; BC = blz.3 vh PSLOT
E287   21C1FC          LD   HL,EXPTBL      ; uitbreidingen
E28A   09              ADD  HL,BC          ;
E28B   CB7E            BIT  7,(HL)         ; PSLOT uitgebreid ?
E28D   280E            JR   Z,AD1          ; Z = normaal
E28F   21C5FC          LD   HL,SLTTBL      ; secundaire registers
E292   09              ADD  HL,BC          ;
E293   7E              LD   A,(HL)         ; A = secundair register
E294   07              RLCA                ;
E295   07              RLCA                ;
E296   07              RLCA                ;
E297   07              RLCA                ;
E298   E60C            AND  0CH            ; A = blz.3 SSLOT
E29A   B1              OR   C              ; voeg blz.3 PSLOT in
E29B   CBFF            SET  7,A            ; A = slot-identif.
E29D   321FF9  AD1:    LD   (CPNT),A       ;
E2A0   C9              RET                 ;

E2A1   00      CHRNUM: DEFB 0              ; huidige karakter
E2A2   00      DOTNUM: DEFB 0              ; huidige pixel
E2A3           CHRTAB: DEFS 2048           ; patronen tot EAA2H
```

INDEX ALGEMEEN

BASIC KEYWORDS INDEX

ABS	2E82H
ASC	680BH
ATN	2A14H
ATTR$	7C43H
AUTO	49B5H
BASE	7B5AH
BEEP	00C0H
BIN$	65FFH
BLOAD	6EC6H
BSAVE	6E92H
CALL	55A8H
CDBL	303AH
CHR$	681BH
CINT	2F8AH
CIRCLE	5B11H
CLEAR	64AFH
CLOAD	703FH
CLOAD?	703FH
CLOSE	6C14H
CLS	00C3H
CMD	7C34H
COLOR	7980H
CONT	6424H
COPY	7C2FH
COS	2993H
CSAVE	6FB7H
CSNG	2FB2H
CVD	7C70H
CVI	7C66H
CVS	7C6BH
DATA	485BH
DEF	501DH
DEF FN	501DH
DEFDBL	4721H
DEFINT	471BH
DEFSNG	471EH
DEFSTR	4718H
DEFUSR	500EH
DELETE	53E2H
DIM	5E9FH
DRAW	5D6EH
DSKF	7C39H
DSKI$	7C3EH
DSKO$	7C16H
ELSE	485DH
END	63EAH
EOF	6D25H
ERASE	6477H
ERL	4E0BH
ERR	4DFDH
ERROR	49AAH
EXP	2B4AH
FIELD	7C52H
FILES	6C2FH
FIX	30BEH
FN	501DH
FOR	4524H
FPOS	6D39H
FRE	69F2H
GET	775BH
GOSUB	47B2H
GOTO	47E8H
HEX$	65FAH
IF	49E5H
INKEY$	7347H
INP	4001H
INPUT	4B6CH
INPUT$	6C87H
INSTR	68EBH
INT	30CFH
INTERVAL	77B1H
INTERVAL OFF	77B1H
INTERVAL ON	77B1H
INTERVAL STOP	77B1H
IPL	7CA2H
KEY	786CH
KEY(n) OFF	77D4H
KEY(n) ON	77D4H
KEY(n) STOP	77D4H
KILL	7C25H
LEFT$	6861H
LEN	67FFH
LET	4880H
LFILES	6C2AH
LINE	4B0EH
LINE INPUT	4B0EH
LIST	522EH
LLIST	5229H
LOAD	6B5DH
LOC	6D03H
LOCATE	7766H
LOF	6D14H
LOG	2A72H
LPOS	4FC7H
LPRINT	4A1DH
LSET	7C48H
MAXFILES	7E4BH
MERGE	6B5EH
MID$	689AH
MKD$	7C61H
MKI$	7C57H
MKS$	7C5CH
MOD	323AH
MOTOR	73B7H

BASIC KEYWORDS INDEX

ROM BIOS routines INDEX

BEEP	00C0H	> 1113H
BREAKX	00B7H	> 046FH
CALATR	0087H	> 06F9H
CALBAS	0159H	> 01FFH
CALLF	0030H	> 0205H
CALPAT	0084H	> 06E4H
CALSLT	001CH	> 0217H
CHGCAP	0132H	> 0F3DH
CHGCLR	0062H	> 07F7H
CHGET	009FH	> 10CBH
CHGMOD	005FH	> 084FH
CHGSND	0135H	> 0F7AH
CHKRAM	0000H	> 02D7H
CHPUT	00A2H	> 08BCH
CHRGTR	0010H	> 2686H
CHSNS	009CH	> 0D6AH
CKCNTC	00BDH	> 10F9H
CLRSPR	0069H	> 06A8H
CLS	00C3H	> 0848H
CNVCHR	00ABH	> 089DH
DCOMPR	0020H	> 146AH
DISSCR	0041H	> 0577H
DOWNC	0108H	> 172AH
DSPFNK	00CFH	> 0B2BH
ENASCR	0044H	> 0570H
ENASLT	0024H	> 025EH
ERAFNK	00CCH	> 0B15H
FETCHC	0114H	> 1639H
FILVRM	0056H	> 0815H
FNKSB	00C9H	> 0B26H
FORMAT	0147H	> 148EH
GETVCP	0150H	> 1470H
GETVC2	0153H	> 1474H
GETYPR	0028H	> 2689H
GICINI	0090H	> 04BDH
GRPPRT	008DH	> 1510H
GSPSIZ	008AH	> 0704H
GTASPC	0126H	> 18C7H
GTPAD	00DBH	> 12ACH
GTSTCK	00D5H	> 11EEH
GTTRIG	00D8H	> 1253H
GTPDL	00DEH	> 1273H
INIFNK	003EH	> 139DH
INIGRP	0072H	> 05D2H
INIMLT	0075H	> 061FH
INIT32	006FH	> 0538H
INITIO	003BH	> 049DH
INITXT	006CH	> 050EH
INLIN	00B1H	> 23D5H
ISCNTC	00BAH	> 03FBH
ISFLIO	014AH	> 145FH
KEYINT	0038H	> 0C3CH
KILBUF	0156H	> 0468H
LEFTC	00FFH	> 16EEH
LFTQ	00F6H	> 14EBH
LDIRMV	0059H	> 0744H
LDIRVM	005CH	> 070FH
LPTOUT	00A5H	> 085DH
LPTSTT	00A8H	> 0884H
MAPXYC	0111H	> 15DFH
NMI	0066H	> 1398H
NSETCX	0123H	> 1809H
OUTDLP	014DH	> 1B63H
OUTDO	0018H	> 1B45H
PHYDIO	0144H	> 148AH
PINLIN	00AEH	> 23BFH
PNTINI	0129H	> 18CFH
POSIT	00C6H	> 088EH
PUTQ	00F9H	> 1492H
QINLIN	00B4H	> 23CCH
RDPSG	0096H	> 110EH
RDSLT	000CH	> 01B6H
RDVDP	013EH	> 1449H
RDVRM	004AH	> 07D7H
READC	011DH	> 1647H
RIGHTC	00FCH	> 16C5H
RSLREG	0138H	> 144CH
SCALXY	010EH	> 1599H
SCANL	012FH	> 197AH
SCANR	012CH	> 18E4H
SETATR	011AH	> 1676H
SETC	0120H	> 167EH
SETGRP	007EH	> 0602H
SETMLT	0081H	> 0659H
SETRD	0050H	> 07ECH
SETT32	007BH	> 05B4H
SETTXT	0078H	> 0594H
SETWRT	0053H	> 07DFH
SNSMAT	0141H	> 1452H
STMOTR	00F3H	> 1384H
STOREC	0117H	> 1640H
STRTMS	0099H	> 11C4H
SYNCHR	0008H	> 2683H
TAPIN	00E4H	> 1ABCH
TAPIOF	00E7H	> 19E9H
TAPION	00E1H	> 1A63H
TAPOOF	00F0H	> 19DDH
TAPOON	00EAH	> 19F1H
TAPOUT	00EDH	> 1A19H

ROM-BIOS routines INDEX

SYSTEEMVARIABELEN INDEX

INSFLG	FCA8H
INTCNT	FCA2H
INTFLG	FC9BH
INTVAL	FCA0H
JIFFY	FC9EH
KANAMD	FCADH
KANAST	FCACH
KBFMIN	F41EH
KBUF	F41FH
KEYBUF	FBF0H
LFPROG	F954H
LINL32	F3AFH
LINL40	F3AEH
LINLEN	F3B0H
LINTTB	FBB2H
LINWRK	FC18H
LOHADR	F94BH
LOHCNT	F94DH
LOHDIR	F94AH
LOHMSK	F949H
LOWLIM	FCA4H
LPTPOS	F415H
MAXDEL	F92FH
MAXFIL	F85FH
MAXUPD	F3ECH
MCLFLG	F958H
MCLLEN	FB3BH
MCLPTR	FB3CH
MCLTAB	F956H
MEMSIZ	F672H
MINDEL	F92DH
MINUPD	F3EFH
MLTATR	F3D7H
MLTCGP	F3D5H
MLTCOL	F3D3H
MLTNAM	F3D1H
MLTPAT	F3D9H
MOVCNT	F951H
MUSICF	FB3FH
NAMBAS	F922H
NEWKEY	FBE5H
NLONLY	F87CH
NOFUNS	F7B7H
NTMSXP	F417H
NULBUF	F862H
OLDKEY	FBDAH
OLDLIN	F6BEH
OLDSCR	FCB0H
OLDTXT	F6C0H
ONEFLG	F6BBH
ONELIN	F6B9H
ONGSBF	FBD8H

PADX	FC9DH
PADY	FC9CH
PARM1	F6E8H
PARM2	F750H
PATBAS	F926H
PATWRK	FC40H
PDIREC	F953H
PLYCNT	FB40H
PRMFLG	F7B4H
PRMLIN	F6E6H
PRMLN2	F74EH
PRMPRV	F74CH
PRMSTK	F6E4H
PROCNM	FD89H
PRSCNT	FB35H
PRTFLG	F416H
PTRFIL	F864H
PTRFLG	F6A9H
PUTPNT	F3F8H
QUEBAK	F971H
QUETAB	F959H
QUEUEN	FB3EH
QUEUES	F3F3H
RAWPRT	F418H
RDPRIM	F380H
REPCNT	F3F7H
RG0SAV	F3DFH
RG1SAV	F3E0H
RG2SAV	F3E1H
RG3SAV	F3E2H
RG4SAV	F3E3H
RG5SAV	F3E4H
RG6SAV	F3E5H
RG7SAV	F3E6H
RNDX	F857H
RS2IQ	FAF5H
RTPROG	F955H
RTYCNT	FC9AH
RUNBNF	FCBEH
SAVEND	F87DH
SAVENT	FCBFH
SAVSP	FB36H
SAVSTK	F6B1H
SAVTXT	F6AFH
SAVVOL	FB39H
SCNCNT	F3F6H
SCRMOD	FCAFH
SKPCNT	F94FH
SLTATR	FCC9H
SLTTBL	FCC5H
SLTWRK	FD09H

STATFL	F3E7H
STKTOP	F674H
STREND	F6C6H
SUBFLG	F6A5H
SWPTMP	F7BCH
T32ATR	F3C3H
T32CGP	F3C1H
T32COL	F3BFH
T32NAM	F3BDH
T32PAT	F3C5H
TEMP	F6A7H
TEMP2	F6BCH
TEMP3	F69DH
TEMP8	F69FH
TEMP9	F7B8H
TEMPPT	F678H
TEMPST	F67AH
TRCFLG	F7C4H
TRGFLG	F3E8H
TRPTBL	FC4CH
TTYPOS	F661H
TXTATR	F3B9H
TXTCGP	F3B7H
TXTCOL	F3B5H
TXTNAM	F3B3H
TXTPAT	F3BBH
TXTTAB	F676H
USRTAB	F39AH
VALTYP	F663H
VARTAB	F6C2H
VCBA	FB41H
VCBB	FB66H
VCBC	FB8BH
VLZADR	F419H
VLZDAT	F41BH
VOICAQ	F975H
VOICBQ	F9F5H
VOICCQ	FA75H
VOICEN	FB38H
WINWID	FCA5H
WRPRIM	F385H
WRPRM1	F388H

W.R. van der Ent
Ogierssingel 281
3076 CT ROTTERDAM
Tel. 010-4329950

ERRATUM

Op blz.245 is bij het drukken de tekst bij adres FC9CH weggevallen.

Hier moet staan:

FC9CH PADY:DEFB 00H

Deze variabele bevat het Y-coordinaat van het laatste punt dat door een electronisch tekenbord werd gedetecteerd.

TERMINAL

HET MSX ROM-BIOS HANDBOEK

Dit handboek vertelt U alles over de MSX Standaard en is onontbeerlijk voor de gevorderde gebruiker, die ten volle gebruik wil maken van alle mogelijkheden, die de MSX te bieden heeft.

Begonnen wordt er met een complete bespreking van het MSX Systeem en de specificaties van de PPI (Programmeerbare Periferie Interface), de VDP (Video Display Processor) en de PSG (Programmeerbare Sound Generator).
Zowel de Hardware als de Software komen ter sprake.

Dit wordt gevolgd door een zeer uitgebreide analyse van de MSX-ROM; het BIOS gedeelte dat het Operating System bevat en de BASIC-INTERPRETER die het Microsoft Extended Basic bevat.

Verder een uitgebreide bespreking van de SYSTEEMvariabelen en de zgn. HOOKS die door het BIOS worden gebruikt.

Het handboek wordt afgesloten met een aantal programma's die gebruik maken van de informatie in dit handboek.

Een absolute must voor alle serieuze MSX gebruikers.

TERMINAL SOFTWARE PUBLICATIES